DOCUMENT RESUME

ED 176 998 SE 029 046

AUTHOR Conklin, James W.; Barnett, Mark
TITLE A Basic Software Package for the Computek Terminal on MTS.
INSTITUTION Michigan Univ., Ann Arbor. Center for Research on Learning and Teaching.
SPONS AGENCY Air Force Office of Scientific Research, Washington, D.C.; National Science Foundation, Washington, D.C.4;
PUB DATE Aug 71
GRANT AFOSR-68-1601; NSF-GZ-2975
NOTE 149p.; Page 2.5.1 missing from document prior to its being shipped to EDRS for filming; Best copy available

EDRS PRICE MF01/PC06 Plus Postage.
DESCRIPTORS Computer Graphics; *Computer Programs; *Computers; *Manuals; *Programers; Programing
IDENTIFIERS *Graphic Display; *Software

ABSTRACT

This material is intended as a programmer's reference for the software package contained in the MTS file K033:CKLIB for use with a Computek terminal. It is assumed that the reader is familiar with FORTRAN programing in MTS and is acquainted with the Computek terminal. (MK)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U S DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Mary L. Charles
NSF
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

ED176998

Center for Research on
Learning and Teaching

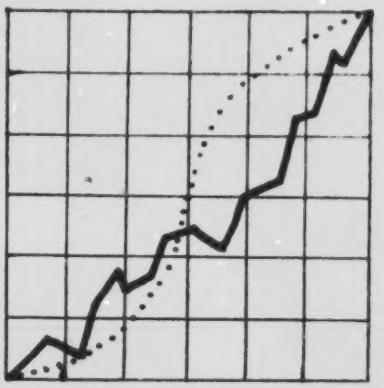
The University of Michigan

August 1971

A Basic Software Package for the Computek Terminal on MTS

James W. Conklin

Mark Barnett



Copyright © The University of Michigan, 1971.

SE 029 046

Preface

The Center for Research on Learning and Teaching has a COMPUTEK Model 400/20 graphic-display terminal, one of several similar terminals which are owned, on order or being considered by various University units and departments. Although the terminal can be used on MTS in a manner similar to an ordinary Teletype, its graphic-display facilities require the use of additional software for support of its particular I/O-processing characteristics. That software typically takes the form of a package of subroutines whose object modules are stored in a file (for example, K033:CKLIB); the user obtains graphical output, then, by concatenating the subroutine file to his program file when issuing the RUN command:

```
$RUN MYPROG+K033:CKLIB . . .
```

Initial work on the Computek was accomplished by using a subroutine package provided by the manufacturer. Though designed to permit full use of the machine's capabilities, the package had two major shortcomings with respect to general and continued use: (a) relatively high processing cost (it was coded as a highly modular set of FORTRAN subroutines); and, (b) relatively detailed and complex programming required for its use. It was felt that advantages in cost and convenience would justify the re-writing of the basic software; as a consequence, the package described here has been produced as an alternative to that of the manufacturer.

This writeup is intended as a programmer's reference for the software package contained in the MTS file K033:CKLIB. It is assumed that the reader is familiar with FORTRAN programming in MTS and has at least a nodding acquaintance with the COMPUTEK terminal. For additional information, see the reference sources listed in Appendix A.

Questions and comments may be directed to James W. Conklin, Center for Research on Learning and Teaching, 109 East Madison St., Ann Arbor.

James W. Conklin
Mark Barnett

This work was supported in part by a Grant from the Air Force Office of Scientific Research (AFOSR-68-1601).

Table of Contents

Preface

Table of Contents

Introduction - 1.1
Organization and Use of the Manual - - - - - - - - - - 1.1
Basic Operation of the Terminal- - - - - - - - - - - - 1.1
Local Control- 1.3
Program Control- - - - - - - - - - - - - - - - - - - 1.5
A Computek Display - - - - - - - - - - - - - - - - - - 1.6
Display Construction - - - - - - - - - - - - - - - - - 1.7
Display Management - - - - - - - - - - - - - - - - - - 1.11
Input Control- 1.12
Error Processing - - - - - - - - - - - - - - - - - - - 1.13

Subroutine Descriptions- - - - - - - - - - - - - - - - - - 2.0
Mnemonics, Conventions and Notation- - - - - - - - - - 2.1
Basic Graphics - 2.2.0
CKER - 2.2.1
CKDER- 2.2.2
CKMA - 2.2.3
CKVA - 2.2.4
CKVABS - 2.2.5
CKMI - 2.2.6
CKVI - 2.2.7
CKVINC - 2.2.8
CKMS - 2.2.9
CKVS - 2.2.10
CKSHOR - 2.2.11
CKML - 2.2.12
CKVL - 2.2.13
CKLONG - 2.2.14
CKCA - 2.2.15
CKCI - 2.2.16
CKCS - 2.2.17
CKCSA- 2.2.18
CKCSI- 2.2.19
CKCUSA - 2.2.20
CKSB - 2.2.21

Character and Textual I/O- - - - - - - - - - - - - - - - 2.3.0
CKTXT- 2.3.1
CKCVT- 2.3.2
CKCERR - 2.3.3
CKNEXI - 2.3.4

Curve and Vector Fitting - - - - - - - - - - - - - - - - 2.4.0
CKVFIT - 2.4.1
CKCFIT - 2.4.2
CKSMTH - 2.4.3

Special Figures- 2.5.0
CKCIR- 2.5.1
CKARCS - 2.5.2
CKARC- 2.5.3
CKRECT - 2.5.4
CKDASH - 2.5.5

Region Management- - - - - - - - - - - - - - - - - - - 2.6.0
CKREG- 2.6.1
CKORG- 2.6.2
CKSCA- 2.6.3
CKGRIN - 2.6.4
CKACT- 2.6.5
CKIACT - 2.6.6
CKDREG - 2.6.7

Windowing- 2.7.0
CKVABW - 2.7.1

Buffer Management- - - - - - - - - - - - - - - - - - - 2.8.0
CKTRAN - 2.8.1
CKTRH- 2.8.2
CKPURG - 2.8.3
CKBLEN - 2.8.4
CKSAVE - 2.8.5
CKRSTO - 2.8.6
CKOPUT - 2.8.7

Object Definition and Use- - - - - - - - - - - - - - - 2.9.0
CKOFIL - 2.9.1
CKCOB- 2.9.2
CKSOB- 2.9.3
CKGOB- 2.9.4
CKIOB- 2.9.5
CKDOB- 2.9.6
CKLOB- 2.9.7

Input Processing - - - - - - - - - - - - - - - - - - - 2.10.0
CKRC - 2.10.1
CKRCRT - 2.10.2
CKFBTN - 2.10.3
CKABTN - 2.10.4
CKATTN - 2.10.5

Sample Programs and Displays - - - - - - - - - - - - - - 3.1

Appendices - 4.0
A: Additional References - - - - - - - - - - - - - - - 4.1
B: Basic Operation of the Computek Terminal- - - - - - 4.2
C: Curve Equations - - - - - - - - - - - - - - - - - - 4.3
D: Updating Procedures for Manual and Library- - - - - 4.4
E: ASCII Characters and Numeric Equivalents- - - - - - 4.5
F: Shape and Placement of Printing Characters- - - - - 4.6

Index- 5.0

Introduction

The basic assumption underlying the design of this software package is that the "typical user" of a Computek is less interested in the physical characteristics of the device than in its effective and efficient use. The basic software, then, should reflect the user's requirements for display design, rather than the data requirements of the display terminal. Although the ideal of "hardware transparency" cannot be achieved completely, this package attempts to embody that concept to the extent permitted by the Computek's design. Further software development, as dictated by the needs of individual users, can be based on this package and can enhance further the ease with which his particular displays can be generated and manipulated.

Organization and Use of the Manual

The major portion of the manual consists of documentation for the library of subroutines located in the permitted MTS file K033:CKLIB. The set of subroutine descriptions is organized into subsets, according to the function performed by the individual routines. That arrangement, like the organization of the manual itself, was chosen for the convenience of the reader who needs a "user's guide" as well as a reference manual. (An alphabetical listing of the subroutines will be found in the Index.) After becoming familiar with the information in this introductory section and with the general contents of the remaining sections, the reader should be able to use the relatively detailed table of contents, the index, and the internal cross-references to locate readily the information desired. Of particular importance is the section on "Mnemonics, Conventions and Notation", which contains helpful and necessary information on the understanding and use of the various routines.

Basic Operation of the Terminal

Operation of a Computek on MTS is initiated in a manner quite similar to that used with an ordinary Teletype or other printing terminal (see Appendix B). In fact, a Computek may be used in place of a printing terminal, provided a hard copy (terminal listing) of the session is not required; however, even that restriction may be removed if a separate, hard-copy device is attached to the basic terminal unit.

When being used strictly as a printing terminal, the Computek operates entirely in its character mode; that is, information received from MTS or entered from the keyboard is interpreted and displayed in character form. (Figure 1 reproduces the hard copy made from such a terminal session.) No special software need be referenced for such use, since the sign-on procedure establishes the required conventions with MTS and the Data Concentrator. Most applications of the Computek will involve both character mode and graphic mode. In the latter mode, information is

PPN : ANN ARBER (DC60-6666)
@dlg k066 pwd
@**LAST SIGNON WAS: 10:37.47 08-09-71
USER 'K066' SIGNED ON AT 10:38.21 ON 08-09-71
@copy file
>THIS SHORT FILE IS BEING COPIED TO ILLUSTRATE THE APPEARANCE AND
>DISTRIBUTION OF CHARACTER OUTPUT ON THE COMPUTEK SCREEN. NOTE THAT
>AFTER THE COMMAND '$SOURCE K033:SETUP' IS ENTERED. THE LINE WHICH
>FOLLOWS THIS ONE IS NOT TRUNCATED AT THE TELETYPE LENGTH OF 72
>CHARACTERS. ##########
@source k033:setup
#$RT ECHO-OFF
#$SOURCE PREVIOUS
#list file
> 1 THIS SHORT FILE IS BEING COPIED TO ILLUSTRATE THE APPEARANCE AND
> 2 DISTRIBUTION OF CHARACTER OUTPUT ON THE COMPUTEK SCREEN. NOTE THAT
> 3 AFTER THE COMMAND '$SOURCE K033:SETUP' IS ENTERED. THE LINE WHICH
> 4 FOLLOWS THIS ONE IS NOT TRUNCATED AT THE TELETYPE LENGTH OF 72
> 5 CHARACTERS. ##########
END OF FILE
@run *catalog perm3.disk
@EXECUTION BEGINS
FBTHDR CKTREE TREE
TREEOBJ PSYCH TEMP
USER K066 HAS 6 FILE(S) WITH TOTAL SIZE OF 12 DISK PAGES
@EXECUTION TERMINATED
$comment I just moved the cursor over and down locally for the
insertion of these characters. The system sees
this message as one long input line.
@$ FILE 'LONG' DOES NOT EXIST.
@comment I don't know where that message came from!!!
@signoff $
@OFF AT 10:43.32 08-09-71
$.39
$175.06
R

Figure 1

interpreted and displayed as a sequence of commands which result in the drawing of figures composed of straight lines and curves. Graphic display is usually handled with special software such as CKLIB, the package described here. (The problem of "mode" is crucial--identical output produces different results in different terminal modes--and complex--there are really several graphic modes, and the mode itself is determined by output received from the system. The mode of the terminal need not be a concern in most cases, however, since CKLIB will manage the problem so that mode changes are transparent to the user.)

Visible images--characters and lines--are "written" on the screen by an electron gun whose current "target" (screen position) is indicated by the cursor, a visible-but-non-printing pointer shaped thus: ⌐ . The physical location of the cursor is always some point on a 1024x1024 square grid, only 1024x800 points of which are visible on the Computek's rectangular screen. (See Fig. 2.) Attempting to write past the right or left edges of the grid will result in wraparound to the opposite edge. Similar wraparound occurs at the top and bottom edges of the grid (not the screen); therefore, visual information is lost while the cursor is located in the top portion of the grid.

Images are displayed on the screen by means of a hardware process which stores the graphic image in its visible form only; i.e., the information (bits) which the terminal processes to produce a character or line on the screen is processed only once--when the image is first drawn on the screen. Again, the similarity with a printing terminal is clear. An awkward by-product of that hardware feature dictates that the screen image be erased completely or not at all: erasing any portion of the display implies erasing the entire display.

Thus, there are three features of the terminal--mode-setting, cursor-positioning and screen-erasure--which have no direct counterparts for printing terminals. These features are controlled both locally, by the user at the keyboard, and by output from the user's program. (Important: In the following discussion and throughout the manual, unless stated otherwise, "local control" will refer to action controlled by the keyboard while the terminal is on-line; for "local mode"--operating the terminal as a stand-alone device--see the User's Manual.)

Local Control. As shown in Figure 3, the terminal keyboard contains a standard character set plus three clusters of auxiliary keys. The six function keys each transmit a non-printing character and are frequently used as selector switches by application programs. In addition, some of those keys also change the mode (character vs. graphic) of the terminal. Since CKLIB was designed to make mode-setting transparent to the user and the programmer, no more will be said here than the caution to refrain from using the function keys unless their use is specifically provided for by the program being written or being run (see below in the "Program Control" section).

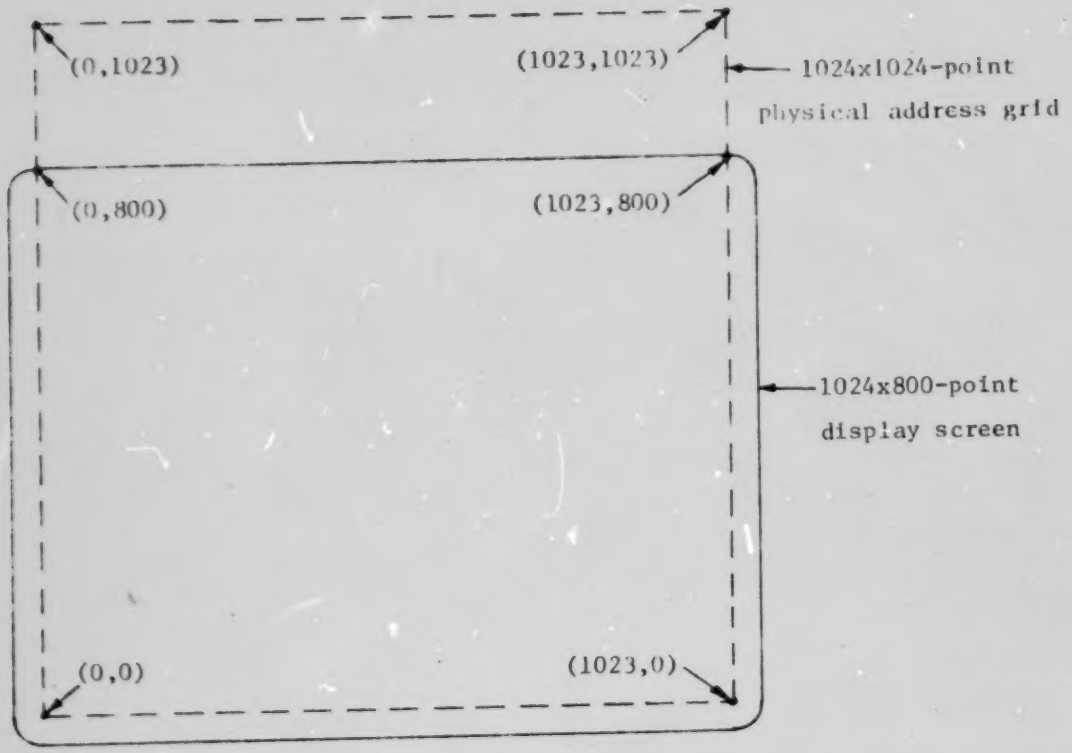


Figure 2

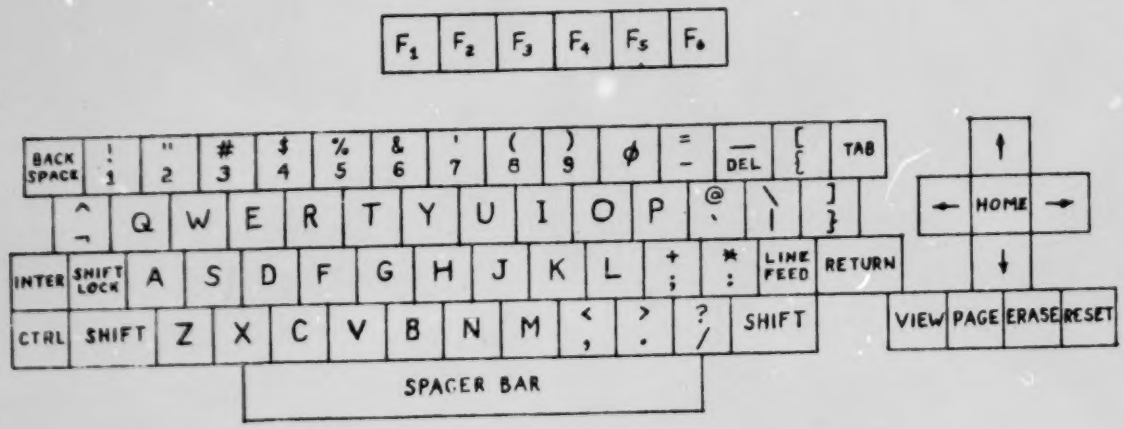


Figure 3

The cross-shaped cluster of keys permits the user to position the cursor locally by moving it in any of the four indicated directions or by pressing the HOME key, which positions it at the upper-left character position on the screen. These five keys do not transmit any information to the system.

The four keys at the lower-right of the keyboard operate as follows: the VIEW key--no character transmission--is used to brighten a screen image which the terminal has automatically dimmed to prevent physical damage to the screen (lack of transmission for 20 seconds will result in automatic dimming of the screen image); the ERASE key--no character transmission--erases the entire screen image locally; the RESET key--no character transmission--sets the cursor to its "home" position (see above) and places the terminal in its character mode; the PAGE key transmits to the system a single character which, when echoed by the Data Concentrator, produces the effect of an ERASE and a HOME.

Program Control. Since this description is confined to on-line use, the terminal would always be under program control in the sense that some programs--the Data Concentrator, MTS, CKLIB or others--would have functioning links with the terminal. A brief overview of system control vs. program control may be helpful for providing perspective, although details and special cases must be found in other sources and will become familiar only through actual use.

To the system the Computek appears to be a printing terminal. Carriage returns, line feeds, etc., are handled appropriately under system control, as they are with other devices; however, mode-setting, cursor-positioning and screen-erasure--features particular to the Computek--must be handled under program control, not system control. Conceptually, the system serves as a transparent "window" through which these special controls are transmitted from the program to the terminal. A similar situation holds with graphical output (i.e., output intended for graphical interpretation while the terminal is in one of its non-character modes): the user must provide for any problems associated with translating information between its external and internal forms. A simple example should clarify the situation and should indicate the necessity for the kind of software support provided by CKLIB.

Consider a terminal session consisting of first the signon, then the copying (to the terminal) of a graphic display file containing both lines and characters, then the signoff. Signon and signoff are handled by the system, which automatically performs the required translations. Copying of the display file, however, requires that no translation be performed by the system, since graphic commands are expressed in binary form. (Such a command might be $COPY DISPLAY *SINK*@BIN, where "@BIN" specifies binary transmission.) Ensuring that the figures and characters in the display "look right" when copied, then, becomes the task of the program which was used in constructing the display file originally. Suffice it to say that this task, though anything but trivial, may be performed

rather easily if the programmer has access to a software package which can keep track of the bookkeeping, translation schemes, etc. which are required. That, basically, is the function of CKLIB.

When the term "program control" is used with reference to the Computek, then, the contrast with "system control" is somewhat sharper than it is with printing terminals. There are more things going on, and fewer of them are under the direct influence of MTS. Mode-setting is handled transparently by CKLIB, which "knows" which mode to use by keeping track of what the user and the program have been doing. Positioning of the cursor--as such--is almost forgotten. The programmer is concerned with designing displays; the cursor becomes his writing instrument, much as a piece of chalk is used to produce a blackboard drawing. Erasing the screen, again under program control, is equivalent to erasing the board for a new display.

A Computek Display

The term display will refer to the visual content of the terminal screen at a particular time. A display consists of whatever has been drawn since the screen was last erased and until the next erasure. (Figure 1 is an example of a single display.) Every terminal session will involve one or more displays, then, whether the session involves graphical output or not. In most cases, including the following discussion, "display" will refer in particular to one of the distinct "pictures" which the programmer plans in advance as one unit of a program-controlled sequence. Examples of such displays are given in Section 3.

A completed display may contain various kinds of information from a variety of sources. For example, a program might be written to produce graphs as specified by the terminal user. A complete display might contain: messages from the program requesting parameter values; the text of the user's responses; the resulting graph, with character labels, straight lines and curved lines. The program, then, would be designed to produce a sequence of such displays--erasing each display at the user's request, constructing the next, and so on--until the user wishes to terminate the program.

Generally speaking, an application program designed for the Computek will be organized in terms of a sequence of displays to be built-up and erased in response to the user's inputs. There is a great deal of variation, however, in the uses made of these displays--from high-speed retrieval and output of text, to question-answer dialogs, to graphical simulation of processes, to individual frames of animated motion pictures. Corresponding to this variety of display formats is the need for flexibility and choice in the planning, construction and programming of individual displays and display sequences. The remainder of this introductory section will be devoted to an overview of the facilities

offered by CKLIB for constructing displays (or parts of displays), drawing them to scale, storing them temporarily or permanently, and managing the interaction between terminal and program. Additional information may be found in the brief prefaces which introduce the various categories of subroutine descriptions.

Two considerations deserve mention at this point: (a) Computek programs are not restricted to the use of CKLIB for input and output--other I/O routines (FORTRAN, system, etc.) may be used when appropriate; (b) the various facilities offered by CKLIB are inter-related in their effects, requiring that the programmer learn more about the whole package as his programs become more complex. Simple displays may be programmed simply, with optional features defaulting to standard values. Complex displays require the familiarity which comes only with hands-on use and experimentation.

Display Construction

The set of visual elements which may be used to construct displays is determined by the hardware capabilities of the terminal. We are concerned here with the standard Model 400/20, which can generate characters (of one size only), straight line segments (called vectors) and curved line segments (called curves).

Addressing. The placement of images on the screen is determined, ultimately, with reference to the physical address grid shown in Figure 2. Since the terminal's built-in coordinate system is not the most convenient to use in many cases, CKLIB permits the user to re-define the coordinate system to be used by the program in referring to grid locations. Four terms which appear throughout the manual are defined as follows:

Physical address: an ordered pair, (x,y), where x and y are integers between 0 and 1023 inclusive, which specifies a unique point in the Computek's hardware address grid. The base region is that display area which is described in terms of physical address.

Logical address: an ordered pair, (x,y), of numbers which specifies a point in a display grid (coordinate system). A logical region is a display area which is described in terms of logical addresses. The base region is the only predefined logical region.

If one thinks of the display (or screen) grid in terms of a Cartesian graph, the entire base region lies in the first quadrant, the x and y grid lines assuming integral values from 0 through 1023. The cells of this grid are uniquely referenced by specifying the (x,y) coordinate pair at the cell's lower-left corner. (A potential source of confusion will disappear by noting that lines drawn on the screen have a width, unlike the idealized grid lines. That width is equal to the side of a

cell--1/1024th the width of the screen.) Continuing in the Cartesian graph framework: a logical region may be defined with a scale factor and/or with its origin "shifted" with respect to the base region. Addresses used in the program, then, may retain their "natural" values (for the particular application), CKLIB transparently performing the required transformation which represents those values appropriately on the screen.

In any case, graphic commands are interpreted within CKLIB in terms of the active region, which the program has selected from the set containing the base region and any logical regions which have been previously defined. If no other logical regions have been defined, the base region is the only logical region, by default, and is the active region for all addressing.

Details of region management are presented later (see section 2.6); though not essential to an initial understanding of terminal use, the concept of logical addressing is important enough to warrant its introduction here for consideration in the remainder of the Introduction.

Characters. The terminal's character set consists of upper- and lower-case letters and the special characters indicated in Figure 3. Each character is drawn with respect to a character cell which is 12 physical grid points wide by 20 high. As reflected by Figure 1, the screen will contain 40 lines of 85 characters each. Since character generation is performed by hardware, rather than software, the size of characters is not affected by region scale factors, while their relative positions may be.

As indicated previously, character output may be produced by standard techniques such as FORTRAN WRITE-statements (cf. Example 2 in Section 3). Sophisticated programs, however, will generally require CKTXT to handle problems of character translation. It is suggested that the programmer become familiar with the basic use of CKTXT as soon as possible. A "trick" worth mentioning is that of substituting characters for small graphical figures when appropriate. A small "o" could represent a point on a graph or a small circle, for example, with some saving in programming time, execution time and image quality.

Vectors and cursor movement. Most of the basic graphic commands result in a movement of the cursor (or beam) from one grid position to another. If the path of the cursor is drawn, the move is called visible; if not, invisible. The result of a single visible move is either a straight-line segment, a vector, or a curved-line segment, a curve. Curves are discussed in the following section. An invisible move, or move, results in a re-positioning of the cursor, no line being drawn. Using the chalk-and-blackboard analogy again, the graphic routines are used to move the chalk over the board, drawing lines as desired.

Continuing with the analogy, we can introduce the last major distinction among the drawing routines. When a person is drawing a blackboard display, the successive movements of the chalk are determined by two factors--the position of the chalk with respect to the boundaries of the board and its position with respect to the partial figure which has already been drawn. In general, the first line will be drawn at a chosen location on the board, but succeeding lines will be drawn with greater attention to preceding lines than to board locations (consider the "THINK AHead" phenomenon). To permit the corresponding flexibility in specifying cursor movements, routines are available which perform the movements either by address or by increment. The former move the cursor from its current position to the grid position specified; the latter move it from its current position by the specified distances--increments--in the x and y directions. Positions and increments are specified and interpreted in terms of the currently active region.

Two other features of CKLIB ease the task of making repeated subroutine calls for successive cursor movements. Single calls to routines such as CKVABS and CKVINC can result in a large number of cursor movements. Also, the many features of CKTXT include commands which permit the embedding of cursor movements within strings of character output.

Curves. A decision to employ curves in display construction will require the programmer's judgment in weighing trade-offs among several advantages and disadvantages. A brief description of the current hardware and software capabilities for curve generation will be followed by mention of two such considerations.

As mentioned above, curves are produced when the path of a single cursor movement is traced visibly on the screen. The "target" point of the move may be specified by address or by incremental distance, in a manner similar to vectors. There are restrictions, however, on which point-pairs may be connected by curves and on the shapes of the resulting curves:

(1) the two points to be connected must define a non-degenerate rectangle (i.e., neither the x-coordinates nor the y-coordinates of the points can be equal);

(2) the resulting curve will lie entirely within the imaginary rectangle defined by the two points and, unless the curve happens to degenerate to the vector joining the two points, it will lie entirely above that diagonal or entirely below it;

(3) the curve and its derivative will be monotonic increasing or decreasing throughout the interval (i.e., the curve will have no inflection points).

These restrictions are based on design decisions made by the terminal manufacturers: there is a particular family of curves which can be drawn by the standard terminal hardware, and all curved lines in a display must be made up of one or more curves in that family. A more technical description of curve geometry may be found in Appendix C and examples of curve output in Section 3.

Curves need not be used at all, since vector approximations would be adequate for most applications. In fact, the terminal model described here has curve generation as an "extra" capability not found in other, similar models. The advantages of using curves include the obvious one of closer correspondence to "reality". Accomplishing that worthy goal, however, requires that the programmer become familiar enough with the terminal's built-in family of curves that his displays can be, in fact, better than vector approximations. A second advantage of using curves is that of data compression: an adequate vector approximation to some curves may require up to 10 (or more) times the quantity of graphical output required for the curve. Of course, the loss of information inherent in data compression shows up again in the restriction of possible curves to a single family.

Of the terminal's capabilities, those involving curve generation have been developed least within the current version of CKLIB--least, in terms of what it should be possible to do easily with existing software. Further development is envisioned, and users are encouraged to communicate their accomplishments in this area for possible inclusion in CKLIB (see Appendix D).

Special Figures. In addition to routines such as CKVINC, which permits the construction of a multiple-line display via a single subroutine call, CKLIB contains a small number of routines designed to produce particular visual figures of a simple and general nature. These latter are described in Section 2.5. Another special routine, CKVABW (see Section 2.7), provides a "windowing" capability for vector-generated figures.

With regard to additional special figures--visual figures displayed repeatedly--the programmer has at least four options for obtaining his own particular set of figures:

(a) By re-directing buffer output, he can produce a desired figure (or display) repeatedly without having to have it constructed each time.

(b) By creating, storing and recalling graphic objects in an object file, he can take advantage of a CKLIB "bookkeeping" facility which permits him to reference special figures (or displays) by name.

(c) By writing his own subroutine(s), he can "put together" his own figure(s), with greater generality than with the two previous methods, and produce them at will by concatenating his subroutine file to CKLIB when executing.

(d) By programming a routine which is sufficiently general to warrant inclusion in CKLIB, he can add a special figure capability to the existing library.

Options (a), (b) and (d) are described more fully in Sections 2.8, 2.9 and Appendix D, respectively. The availability and exercise of these options reflect much of the real power available for use in CKLIB.

Display Management

Once the programmer has determined the design of his display sequence and has formed a preliminary conception of how it will be constructed, he can proceed with the more specific details of planning individual figures, partial displays, terminal interactions, etc. Of prime importance in display planning and management is CKLIB's use of its output buffer, which can be envisioned as a long, automatically extending vector of core locations for the temporary storage of output information. Every CKLIB-generated display is produced by output which, at one time or another, has passed at least once through the output buffer; therefore, options available for display management are probably understood most readily by considering: (a) how information may be placed in the buffer; (b) what may be done with the buffer and its contents; and, (c) where those contents may end up.

Filling the buffer. Except for CKDER (direct erase) and CKRSTO (file copying), the image-producing routines insert their output sequentially at the current end of the buffer. In addition to the output of the "single-purpose" graphic routines and of CKTXT, pre-defined graphic objects may be inserted into the buffer by a call to CKIOB, as can an arbitrary vector of storage by a call to CKSB. The buffer, initially empty, will continue to extend as each new block of information is added, until it is emptied by the program.

Transmitting and emptying the buffer. The program can "dispose of" the buffer in three ways; in each case, the entire buffer is treated as a logical unit--whatever is done, is done to the whole buffer. It may be: (a) transmitted, with contents retained; (b) purged--emptied and re-initialized; or, (c) transmitted and then purged immediately. Disposition of the buffer is controlled by the buffer management routines (Section 2.8) and by the value of the transmission parameter (sometimes optional) passed in calls to other routines. (See Section 2.1 for conventions pertaining to those transmission parameters.)

Directing buffer output. Unless specified otherwise by the program, CKLIB output will be transmitted to the terminal (or the printer, during batch processing). In the simplest case, each buffer-filling routine can be called so that the buffer receives the routine's block of output, is transmitted immediately to the terminal and is then purged. Output may, however, be directed to other devices, files, logical I/O units or other (i.e., non-buffer) storage locations. A variety of options are available for determining where output is to go and how it is to be retrieved. Details may be found in Sections 2.8 and 2.9.

The two primary objectives of display management are efficiency and flexibility. It is generally more efficient, for both the program and the programmer, to generate displays constructed of long output strings and to generate them as seldom as possible. Keeping those objectives in mind, the programmer can use the facilities of CKLIB to reduce, practically to an irreducible minimum, the computing costs and the necessarily somewhat tedious programming tasks associated with graphical display.

Input Control

"Normal" terminal input can be processed without reference to any of the CKLIB routines. Formatted input conversion with FORTRAN READ-statements may prove sufficient for some programs and will probably be used to some extent by most programs. CKLIB offers two additional facilities, one providing an extension in capabilities for formatted conversion and the other providing control appropriate for the special input characteristics of the Computek terminal.

CKCVT, CKCERR and CKNEXI (Section 2.3) permit the program to preprocess input strings so that fatal errors (e.g., "FCVTH-INVALID CHARACTER IN NUMERIC FIELD") are avoided, allowing programs to die more gracefully, at worst, but in many cases permitting the program to edit faulty input strings and continue execution as if format requirements had been met. As with FORTRAN FORMAT-statements, effective use of these subroutines will require practice and planning. Basic mechanisms have been provided, however, for a substantial degree of flexibility in formatted input conversion.

Routines described in Section 2.10, Input Processing, handle those terminal inputs which have an effect or significance not found in inputs from printing terminals. Having access to the current position of the cursor is essential to many programs, particularly when the user is expected to position the cursor locally to indicate either a specific screen location or a choice from among items in a list. For reasons indicated earlier, the operating mode of the terminal (graphic or character modes) is both highly dynamic and critical. To maintain optimal user convenience, CKLIB automatically keeps the terminal in character mode as much of the time as

possible, although a program may override this convention by using non-CKLIB output routines. Mode problems which would arise otherwise--when certain function keys were pressed or when an attention interrupt was sent during graphic transmission--may be handled by routines written specifically to minimize such problems. Efforts have been proposed for altering the system interfaces, which require substantial, behind-the-scenes activity by CKLIB to maintain mode control; "glitches" which now occur occasionally may soon be rare or non-existent.

Error Processing

When a program does not behave as the programmer intended, an error has occurred. The most serious errors, from the viewpoint of this manual, are those which arise in the system or in CKLIB itself. Although those sources of error will be removed when possible, there is little doubt that numerous "bugs" will show up and be fixed before CKLIB routines perform exactly as described, every time.

Of somewhat lesser seriousness are program errors, which result from particularly unfortunate combinations of data variables. For example, a program might attempt to draw a single curve between any two arbitrary points, with arbitrary initial and final slopes. Some combinations of values would result in a curve; others would cause an error return, which the program could provide for; others--where the two points determine a vertical or horizontal line--result in a program interrupt (zero-divide), and everything quits. Although it might be possible, in theory, to include complete data error-checking in the CKLIB routines--in effect, doing nothing without determining, first, whether it legitimately can be done--the additional programming, the increased overhead in execution cost and the overall degradation of performance do not appear to justify the effort. In general, the programmer is left with the responsibility of ensuring that his programs either do not attempt the impossible or, if they do, take advantage of error returns from the CKLIB routines to handle the problem within the program itself.

Though not errors in a technical sense, but frequent enough to merit mention, are display design errors; some programs are "bad" because they perform as the programmer specified, but not as he intended. That is particularly true of graphic display, in which the limited display area, spatial transformations and the like must be considered in addition to the problems of conventional programming. Again, the "philosophy" of CKLIB is to hold the programmer responsible for his own design. Two features of the terminal, and the way CKLIB handles them, illustrate the effect of this philosophy and indicate its potential advantages for the programmer. In most cases the physical address of a cursor movement is not checked by CKLIB before transmission. If the program sends the cursor--and lines or characters--to that portion of the display grid which lies above the screen and out of sight, so be it; some programmers may

wish to use that area to "absorb" unwanted information (like "#END OF FILE", etc.). If program addresses become "too large" or "too small", high-order bits are chopped off and the cursor is moved to the address which remains; in many instances, "wraparound" provides exactly the effect desired.

When an error occurs, the programmer first should determine whether it was merely an error in design; if so, a minor change in the program should fix it. If not, the insertion of temporary diagnostic statements--statements which print the values of key variables or indicate the occurrence of error returns--may expose program errors. Such errors can often be corrected either by testing values before they are used or by reprogramming to accomplish the same objective by different means. If no source of error can be found in the program, the programmer is encouraged to obtain a listing of his program and to consult either with his instructor or with the person(s) responsible for the maintenance of CKLIB.

Subroutine Descriptions

Object modules of the CKLIB routines are contained in the read-only permitted file K033:CKLIB. Any or all of the routines are made available for program reference--are loaded--by concatenating the file with the object file named in the $RUN command. Concatenation may be done explicitly (e.g., $RUN MYPROG+K033:CKLIB) or implicitly (by adding the file delimiting $CONTINUE WITH K033:CKLIB at the end of the program file). The program may then reference the routines via a CALL statement or, in some cases (cf. CKFBTN), by using the routine name as a function variable.

(Although the internal structure of CKLIB need not concern the programmer, it may be of interest to note that the library consists of several modules, many of them with multiple entry-points; routines containing essentially identical code have been combined for efficiency. Service routines not called by user programs are also contained in the library; these perform such "transparent" tasks as storage management, coordinate transformation, transmission protocol and logical I/O unit assignment.)

The following descriptions of CKLIB routines are grouped according to the roles of the routines in user programs. Although routines within a group are in some sense more closely related to each other than to routines in other groups, there is a substantial degree of interdependence among groups, in that a specific routine's ultimate effect may be modified by routines in several other groups. Each group of descriptions is prefaced by information descriptive of particular features of the group; there is also some indication of the interplay within the group and with other groups.

The section immediately following is essential to a detailed understanding of the subroutine descriptions; the Table of Contents, the Index and the section of Sample Programs and Displays provide additional assistance in locating related information.

Mnemonics, Conventions and Notation

The names of all CKLIB routines are limited to six letters, so they can be called from FORTRAN programs, and start with "CK" (ComputeK) to reduce the possibility of name-conflict with the user's own routines. The remainder of each name is a contraction or mnemonic of a word or words which describe the effect of the routine. Though most of the routines are original with this library, many of the basic routines are similar in effect to routines contained in the terminal manufacturer's subroutine package; to avoid unnecessary confusion, corresponding CKLIB routines retain the manufacturer's mnemonic, prefixed with "CK". The following scheme may be of help in associating basic routine names with their respective functions: Move, Vector, Curve, Address (or Absolute), Increment, Long, Short (for vectors) or Slope (for curves).

Format and syntax of the routine descriptions have been adopted, with minor modifications, from Volume 3 of the MTS Manual. Optional parameters are denoted within brackets; in general, the number and order of optional parameters is not important except for the following constraint: if statement labels are included as error exits, they must be the last parameters in the list; the first such label will be interpreted as the exit for a return code of 4, the second for a return code of 8, and so on. (This latter constraint reflects a requirement of the FORTRAN G-level compiler.) Where a parameter may assume different forms, the acceptable forms are denoted as alternatives, separated by vertical bars (see CKOPUT). If the routines are called from a FORTRAN program, FORTRAN conventions may be used to employ expressions in specifying parameter values.

For programming convenience and flexibility, many of the basic graphic routines accept coordinate and increment values as fullword values (or vectors) of either integer or real type. CKLIB internally inspects the high-order bits of such values to determine their type; because of the differences in internal representation of integer vs. floating-point numbers, appropriate action will be taken for all but extreme (and highly unlikely) values. Routines requiring parameters of a particular type are appropriately documented. Note: This built-in interpretation of coordinate and increment values is logically distinct from and prior to the transformations resulting from the definition of logical regions.

A convention which obtains throughout CKLIB is that of the transmission parameter. That parameter, which appears as required or optional for most of the routines, is a fullword integer which determines whether or not the output buffer is to be transmitted immediately after the particular routine has completed its primary function. Although this parameter would normally be assigned one of the three values 0, 1, or 2, the action taken is determined by:

Value	Action
≤ 0	The output buffer is <u>not</u> transmitted.
1	The buffer is transmitted and immediately purged.
≥ 2	The buffer is transmitted but <u>not</u> purged.

The effect of the transmission parameter, then, will be the same as an immediately succeeding call to one of the routines CKTRAN or CKTRH, for the respective values of 1 and 2-or-greater. <u>IMPORTANT</u>: <u>If the parameter is optional and is not passed, no transmission will occur</u>.

Basic Graphics

This group of routines is highly redundant, in that many of them provide alternative methods of producing the same effects. It is suggested that the programmer who is becoming acquainted with CKLIB look first at CKER, CKDER, CKVABS, CKVINC and CKCUSA. Those general-purpose routines provide most of the basic graphic capabilities. The remaining routines, with the exception of CKSB, produce the same visual images, but do so in a variety of more limited, specialized ways.

CKCS does not produce output for the buffer. Except for CKER and CKDER, the remaining routines have the optional or required transmission parameter which determines whether the output buffer's contents are to be transmitted. Neither CKER nor CKDER transmit the buffer--CKER adds the "erase-home" character to the current end of the buffer, while CKDER transmits the same character directly without affecting the buffer at all. CKVABS, CKVINC and CKCUSA produce output which causes one or more successive cursor movements in straight lines by address, straight lines by increment and curved lines by address and slopes, respectively. The two straight-line routines make each cursor movement visibly (a vector) or invisibly (a move) as determined by the value of that element of a beam vector corresponding to those coordinate-vector elements which specify the target point of the given movement. Details of curve specification may be found in the individual routine descriptions and in Appendix C.

The primary function of this group of routines, of course, is to translate coordinate and curve information into output which will produce the desired effect on the terminal screen. Values passed to these routines are not, however, translated directly into their output form; if they were, the programmer would be forced always to plan in terms of physical addresses and a display screen 1024x800 points in size with origin at the lower-left corner. Before being inserted into the output buffer, coordinate values undergo a linear transformation determined by the origin and scale factor of the currently active region. Since the default transformation--that of the pre-defined base region--is the identity, the beginning programmer may wish to plan in terms of physical addresses and let it go at that. However, the convenience of being able to define new logical regions warrants a close look at the material introducing Section 2.6.

Purpose: To erase the screen and position the cursor at the "home" position.

Calling sequence: CALL CKER

Description: CKER is used for erasing the screen and moving the cursor to the upper left hand corner of the screen. The erase is not done immediately but occurs the next time the output buffer is transmitted.

CKDER

Purpose: To erase the screen and position the cursor at the "home" position.

Calling sequence: CALL CKDFR

Description: CKDER is used for erasing the screen and moving the cursor to the upper left hand corner of the screen. The erase occurs immediately, and the output buffer is not affected in any manner; i.e., this routine does not change the output buffer's contents.

Purpose: Move the cursor invisibly from its present position to the new position specified.

Calling sequence: CALL CKMA(X,Y[,IB])

Parameters:

- X - the x-coordinate of the logical address designating the new position.
- Y - the y-coordinate of the logical address designating the new position.
- IB - if present, is a fullword integer specifying whether transmission is desired.

Description: CKMA is used to position the cursor at the specified logical address. If the active region is the base region and no scale factor has been set, then this routine positions the cursor at the specified physical address.

CKVA

Purpose: Move the cursor visibly from its present position to the new position specified.

Calling sequence: CALL CKVA(X,Y[,IB])

Parameters:

X - the x-coordinate of the logical address designating the new position.
Y - the y-coordinate of the logical address designating the new position.
IB - if present, is a fullword integer specifying whether transmission is desired.

Description: CKVA is used to draw a straight line from the present cursor position to the specified logical address. If the active region is the base region and no scale factor has been set, then this routine draws a line to the specified physical address.

CKVABS

Purpose: To provide a method for moving the cursor either visibly or invisibly n times.

Calling sequence: CALL CKVABS(X,Y,NPT,IBEAM[,IB])

Parameters:

- X - a vector of x-coordinates of the logical addresses designating new positions.
- Y - a vector of y-coordinates of the logical addresses designating new positions.
- NPT - a fullword integer indicating the number of points.
- IBEAM- a fullword integer vector indicating whether to move visibly or invisibly.
- IB - if present, is a fullword integer specifying the type of transmission.

Description: CKVABS takes a vector of x and a vector of y coordinate pairs and moves the cursor from one pair to the next either visibly or invisibly. The cursor is moved from its present position to the first point and is left at the final point upon return. The vector IBEAM is a vector or ones and zeros, a one indicating the move is to be visible and a zero indicating the move is to be invisible. The net effect of this routine is most readily illustrated by the following program segment:

```
      DO 30 I=1,NPT
      J=IBEAM(I)+1
      GOTO(10,20),J
10    CALL CKMA(X(I),Y(I),IB)
      GOTO 30
20    CALL CKVA(X(I),Y(I),IB)
30    CONTINUE
```

Purpose: Moves the cursor invisibly from its present position by the relative distances specified.

Calling sequence: CALL CKMI(X,Y[,IB])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
IB - if present, is a fullword integer specifying whether transmission is desired.

Description: CKMI is used to move the cursor x logically addressable points horizontally and y logically addressable points vertically from the present cursor position. If the active region is the base region and no scale factor has been set, then this routine will move the cursor x physically addressable points and y physically addressable points from the current cursor position.

Purpose: Move the cursor visibly from its present position by the relative distances specified.

Calling sequence: CALL CKVI(X,Y[,IB])

Parameters:

X - the distance to move in the x-direction
Y - the distance to move in the y-direction
IB - if present, is a fullword integer specifying whether transmission is desired.

Description: CKVI is used to draw a straight line from the present cursor location to a new location which is x logically addressable points in the horizontal direction by y logically addressable points in the vertical direction away. If the active region is the base region and no scale factor has been set, then the x and y distances represent physical distances.

CKVINC

Purpose: To provide a method for moving the cursor either visibly or invisibly n times by relative distances.

Calling sequence: CALL CKVINC(X,Y,NPT,IBEAM[,IB])

Parameters:

X	-	a vector of x-direction relative distances.
Y	-	a vector of y-direction relative distances.
NPT	-	a fullword integer indicating the number of points.
IBEAM	-	a fullword integer vector indicating whether to move visibly or invisibly.
IB	-	if present, is a fullword integer specifying the type of transmission..

Description: CKVINC takes a vector of distances which are expressed in terms of their x and y components and moves the cursor by the prescribed relative distances (each distance measurement is relative to the last cursor position) either visibly or invisibly. Distances are expressed in logically addressable points; if the current region is the base region, they also represent physically addressable points. The vector IBEAM is a vector of ones and zeros, a one indicating the move is to be visible and a zero indicating the move is to be invisible. The net effect of this routine is most readily illustrated by the following program segment:

```
      DO 30 I=1,NPT
      J=IBEAM(I)+1
      GOTO(10,20),J
10    CALL CKMI(X(I),Y(I),IB)
      GOTO 30
20    CALL CKVI(X(I),Y(I),IB)
30    CONTINUE
```

Purpose: To move the cursor invisibly from its present position by the relative distances specified.

Calling sequence: CALL CKMS(X,Y[,IB])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
IB - if present, is a fullword integer specifying whether transmission is desired.

Description: CKMS is used in precisely the same fashion as CKMI, the basic difference being that a more rigid limit exists on the relative distances for CKMS. This limitation is that x and y must fall within the range of ±63 physically addressable points; i.e., after all scaling has been done, both x and y must be greater than or equal to -63 and less than or equal to +63. The rationale for this limitation involves the number of bytes of graphical information produced. (It takes four bytes of information for a move using CKMI, but only two bytes of information for a move using CKMS.)

Purpose: To move the cursor visibly from its present position by the relative distances specified.

Calling sequence: CALL CKVS(X,Y[,IB])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
IB - if present, a fullword integer specifying the type of transmission.

Description: CKVS is used in precisely the same fashion as CKVI, the basic difference being that a more rigid limit exists on the relative distances for CKVS. This limitation is that x and y must fall within the range of ±63 physically addressable points; i.e., after all scaling has been done, both x and y must be greater than or equal to -63 and less than or equal to +63. The rationale for this limitation involves the number of bytes of graphical information produced. (It takes four bytes of information for a visible move using CKVI, but only two bytes of information for a visible move using CKVS.)

Purpose: To provide a method for moving the cursor either visibly or invisibly n times by relative distances.

Calling sequence: CALL CKSHOR(X,Y,NPT,IBEAM[,IB])

Parameters:

- X - a vector of x-direction relative distances.
- Y - a vector of y-direction relative distances.
- NPT - a fullword integer specifying the number of points.
- IBEAM - a fullword integer vector indicating whether to move visibly or invisibly.
- IB - if present, a fullword integer specifying the type of transmission.

Description: CKSHOR takes a vector of distances which are expressed in terms of their x and y components and moves the cursor by the prescribed relative distances (each distance measurement is relative to the previous cursor position) either visibly or invisibly. The vector IBEAM is a vector of ones and zeros, a one indicating the move is to be visible and a zero indicating the move is to be invisible. The range limitation to ±63 physically addressable points applies as with CKMS and CKVS. The net effect of this routine is most readily illustrated by the following program segment:

```
      DO 30 I=1,NPT
      J=IBEAM(I)+1
      GOTO(10,20),J
10    CALL CKMS(X(I),Y(I),IB)
      GOTO 30
20    CALL CKVS(X(I),Y(I),IB)
30    CONTINUE
```

Purpose: To move the cursor invisibly from its present position by the relative distances specified.

Calling sequence: CALL CKML(X,Y[,IB])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
IB - if present, is a fullword integer specifying whether transmission is desired.

Description: CKML is used in precisely the same fashion as CKMS. The difference between CKMS and CKML is that in CKMS the distances specified represent single units of distance (i.e., the distance between two adjacent logically addressable points), while in CKML the distances represent 16 units of distance; thus, CKMS(32,32) does exactly the same move as CKML(2,2). The purpose of this routine, then, is to allow maximum distances with minimum information transmission--two bytes per move.

Purpose: To move the cursor visibly from its present position by the relative distances specified.

Calling sequence: CALL CKVL(X,Y[,IB])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
IB - if present, a fullword integer specifying the type of transmission.

Description: CKVL is used in precisely the same fashion as CKVS. The difference between CKVS and CKVL is that in CKVS the distances specified represent single units of distance (i.e., the distance between two adjacent logically addressable points), while in CKVL the distances represent 16 units of distance; thus, CKVS(32,32) draws exactly the same line as CKVL(2,2). The purpose of this routine, then, is to allow maximum distances with minimum information transmission--two bytes per move.

Purpose: To provide a method for moving the cursor either visibly or invisibly n times by relative distances.

Calling sequence: CALL CKLONG(X,Y,NPT,IBEAM[,IB])

Parameters:

X	-	a vector of x-direction relative distances.
Y	-	a vector of y-direction relative distances.
NPT	-	a fullword integer specifying the number of points.
IBEAM	-	a fullword integer vector indicating whether to move visibly or invisibly.
IB	-	if present, a fullword integer specifying the type of transmission.

Description: CKLONG takes a vector of distances which are expressed in terms of their x and y components and moves the cursor by the prescribed relative distances (each distance measurement is relative to the previous cursor position) either visibly or invisibly. The vector IBEAM is a vector of ones and zeros, a one indicating the move is to be visible and a zero indicating the move is to be invisible. The distance transformation, to 16-unit increments of logically addressable points, applies as with CKML and CKVL. The net effect of this routine is most readily illustrated by the following program segment:

```
      DO 30 I=1,NPT
      J=IBEAM(I)+1
      GOTO(10,20),J
10    CALL CKML(X(I),Y(I),IB)
      GOTO 30
20    CALL CKVL(X(I),Y(I),IB)
30    CONTINUE
```

Purpose: To move the cursor visibly through a curve from its current position to the new position specified.

Calling sequence: CALL CKCA(X,Y,IA,IB[,IT])

Parameters:

- X - the x-coordinate of the logical address designating the new position.
- Y - the y-coordinate of the logical address designating the new position.
- IA - a fullword integer specifying the a and a_s curvature parameters.
  $-32 \leq IA \leq 32$
- IB - a fullword integer specifying the b curvature parameter.
  $0 \leq IB \leq 31$
- IT - if present, a fullword integer specifying the type of transmission.

Description: CKCA is used to move the cursor through a curve from its current position to the logical address specified. The curve is drawn using the curvature equations and the curvature parameters. The curvature parameters passed are interpreted in the following manner:

$a_s = 1$ if $IA < 0$, otherwise $a_s = 0$

$a = |IA| - 1, \quad 0 \leq a \leq 31$

$b = IB$

Purpose: To move the cursor through a curve from its current position to a new position which is the specified relative distance from the current position.

Calling sequence: CALL CKCI(X,Y,IA,IB[,IT])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
IA - a fullword integer specifying the a and a_s curvature parameter.
$-32 \leq IA \leq 32$
IB - a fullword integer specifying the b curvature parameter.
$0 \leq IB \leq 31$
IT - if present, a fullword integer specifying the type of transmission.

Description: CKCI is used to move the cursor visibly through a curve from its current position, call it (CX,CY) to the position (CX+X,CY+Y). Similarly to CKMI, X and Y increments are given in logically addressable points. The curve is drawn using the curvature equations and the curvature parameters. The curvature parameters are interpreted in the following manner:

$a_s = 1$ if $IA < 0$, otherwise $a_s = 0$

$a = |IA| - 1, \quad 0 \leq a \leq 31$

$b = IB$

Purpose: To calculate curvature parameters from normalized slopes.

Calling sequence: CALL CKCS(SI,SF,IA,IB[,&rc4])

Parameters:

SI - a real*4 positive normalized initial slope.
SF - a real*4 positive normalized final slope.
IA - a fullword integer location provided for the computed value of the combined a and a_s curvature parameters.
IB - a fullword integer location provided for the computed b curvature parameter.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - curvature parameters cannot be computed.

Description: CKCS takes the two normalized slopes and computes from them the three curvature parameters required in a call to CKCA or CKCI. The normalized slopes are defined in the following manner:

$$SI = \frac{S_i}{\Delta Y / \Delta X}$$

$$SF = \frac{S_f}{\Delta Y / \Delta X}$$

where S_i is the absolute value of the tangent to the curve at the initial point, S_f is the absolute value of the tangent to the curve at the final point, ΔY is the y-component of the distance between the initial and final points (i.e., $\Delta Y = |y_i - y_f|$), and ΔX is the x-component of the distance between the initial and final points (i.e., $\Delta X = |x_i - x_f|$).

Purpose: To move the cursor visibly through a curve from its current position to the new position specified.

Calling sequence: CALL CKCSA(X,Y,SI,SF[,IT][,&rc4])

Parameters:

- X - the x-coordinate of the logical address designating the new position.
- Y - the y-coordinate of the logical address designating the new position.
- SI - a real*4 positive normalized initial slope.
- SF - a real*4 positive normalized final slope.
- IT - if present, a fullword integer specifying the type of transmission.
- rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

- 0 - normal return
- 4 - unable to draw curve because curvature parameters cannot be computed.

Description: CKCSA is used to draw a curve from the current cursor position to the logical address specified. The curve is drawn using the normalized initial and final slopes to calculate the curvature parameters. The net effect of this routine is most readily illustrated by the following program segment:

```
CALL CKCS(SI,SF,IA,IB)
CALL CKCA(X,Y,IA,IB,IT)
```

Purpose: To move the cursor visibly through a curve from its current position to a new position which is the specified relative distance from the current position.

Calling sequence: CALL CKCSI(X,Y,SI,SF[,IT][,&rc4])

Parameters:

X - the distance to move in the x-direction.
Y - the distance to move in the y-direction.
SI - a real*4 positive normalized initial slope.
SF - a real*4 positive normalized final slope.
IT - if present, a fullword integer specifying the type of transmission.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - unable to draw curve because curvature parameters cannot be computed

Description: CKCSI is used to draw a curve from the current cursor position, call it (CX,CY), to the position (CX+X,CY+Y). X and Y increments are given in logically addressable points. The curve is drawn using the normalized initial and final slopes to calculate the curvature parameters. The net effect of this routine is most readily illustrated by the following program segment:

```
CALL CKCS(SI,SF,IA,IB)
CALL CKCI(X,Y,IA,IB,IT)
```

Purpose: To generate one or more connected curves, using two points and two slopes for each curve.

Calling sequence: CALL CKCUSA(XVEC,YVEC,N,SIVEC,SFVEC,IB)

Parameters:

XVEC - a real*4 vector containing the x-coordinates of the logical addresses.
YVEC - a real*4 vector containing the y-coordinates of the logical addresses.
N - a fullword integer containing the number of points in XVEC and YVEC.
SIVEC - a real*4 vector containing initial slopes.
SFVEC - a real*4 vector containing final slopes.
IB - a fullword integer specifying the type of transmission. This parameter is not optional.
(Note: The number of initial and final slopes required is N-1; i.e., the dimension of SIVEC and SFVEC is N-1.)

Description: CKCUSA is used to draw a sequence of curves between successive points on the screen, using point positions and initial and final slopes--the slope of the tangent to the curve at the "from" point and the slope of the tangent to the curve at the "to" point--to calculate the curvature. If the slopes are such that the curvature parameters cannot be computed by the curvature equations, then an invisible move is made to the "to" point and the next point pair is processed. Note: whether the curve is drawn or not, the "to" point becomes the "from" point of the next point pair. The initial slope for a curve does not have to equal the final slope for the previous curve, although a smooth curve will impose this restriction. The effect of a call to CKCUSA is the same as repeated calls to CKCSA, except that only a single type of transmission is specified.

Purpose: To place a string of binary information into the output buffer.

Calling sequence: CALL CKSB(BVEC,ICNT[,IB])

Parameters:

BVEC - is either a fullword or a vector of information to be transmitted.
ICNT - a fullword integer containing the number of bytes to be placed into the buffer.
IB - if present, is a fullword integer specifying the type of transmission.

Description: CKSB copies the binary information from BVEC into the output buffer. If the number of bytes to be copied is one, then the fourth byte of BVEC is copied; otherwise, ICNT bytes are copied starting at the first byte of BVEC. The difference between one byte of information and more than one byte of information was provided so that computed binary information could be passed as fullword integers.

Purpose: To provide a method for moving the cursor either visibly or invisibly n times by distances relative to the current cursor position.

Calling sequence: CALL CKLS(X,Y,NPT,IBEAM)

Parameters:

- X - a real*4 vector containing x-distances relative to the current cursor position
- Y - a real*4 vector containing y-distances relative to the current cursor position
- NPT - a fullword integer indicating the number of points
- IBEAM - a fullword integer vector indicating whether to move visibly or invisibly

Description: CKLS takes a vector of distances which are expressed in terms of their x and y components relative to the current cursor position and moves the cursor by the prescribed distances using a short transmission sequence (short in the sense that the number of data bytes is kept to a minimum by using both CKSHOR and CKVINC transmission). The net result is to improve transmission rate on large data bases.

Purpose: To provide a method for moving the cursor either visibly or invisibly n times

Calling sequence: CALL CKLSA(X,Y,NPT,IBEAM[,IB])

Parameters:

- X - a vector of x-coordinates of the logical addresses designating new positions
- Y - a vector of y-coordinates oj the logical addresses designating new positions
- NPT - a fullword integer indicating the number of points
- IBEAM - a fullword integer vector indicating whether to move visibly or invisibly
- IB - if present, is a fullword integer specifying the type of transmission.

Description: CKLSA takes a vector of x and a vector of y coordinate pairs and moves the cursor from one pair to the next either visibly or invisibly. The cursor is moved from its present position to the first point and is left at the final point upon return. The vector IBEAM is a vector of ones and zeros, a one indicating the move is to be visible and a zero indicating the move is to be invisible. This routine functions exactly as CKVABS except it uses CKLS to generate the moves, thus partially optimizing the data transmission.

Character and Textual I/O

Routines in this group were designed primarily to handle three basic difficulties in the Computek/system interface:

(1) the Computek operates with the ASCII character set, while the system operates primarily with the EBCDIC character set;
(2) the Computek display area is two-dimensional, while conventional I/O and transmission protocols are designed for essentially one-dimensional character strings; and,
(3) the Computek's being an interactive terminal places a premium on capabilities for I/O error-recovery, while the heavily used FORTRAN conversion routines were not designed to overlook, or compensate for, improperly formatted I/O.

CKTXT, designed originally to provide EBCDIC to ASCII conversion for the output of fixed character strings, has been developed into a general-purpose output routine. Though its description is somewhat overpowering at first, several examples of its use are given in Section 3 to illustrate many of its features. It will be seen that simple textual output may be programmed quite simply, while the more complex examples will be properly appreciated only after they have been traced through, fully understood and compared with alternative methods of programming the same displays. With CKCVT providing access to the output form of numeric variables, CKTXT offers a workable solution to output aspects of the three problems mentioned above.

CKCVT, CKCERR and CKNEXI, on the other hand, are concerned primarily with the processing of character input and, in particular, with the third of the above-mentioned problems. Although these routines do not have the built-in flexibility and generality of CKTXT--the varieties of "garbage-in" far outnumber those of "garbage-out"--they at least make it possible for the programmer to anticipate potential problems and attempt to deal with them before the program is forced to abandon the conversation. Example 8 in Section 3 illustrates one application of input conversion; another is outlined in the description of CKNEXI. Special-purpose input routines are discussed in Section 2.10.

Purpose: To provide a versatile method of displaying text.

Calling sequence: CALL CKTXT(CSTR,X,Y[,RGOB][,STR][,&rcn])

Parameters:

CSTR - a fullword vector of commands and/or text to be processed and displayed.
X - a fullword vector containing values for x-relative distances and/or x-coordinates.
Y - a fullword vector containing values for y-relative distances and/or y-coordinates.
RGOB - if present, a fullword integer vector containing region names and/or object names. Note: the first word in this vector must be the integer negative one.
STR - if present, a fullword integer vector containing text. Note: the first word in this vector must be the integer one.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - command error
8 - string greater than 255 characters

Description: CKTXT is used to place textual information into the output buffer in ASCII; therefore, graphic and textual data may be "mixed" in a single output buffer. The main purpose of this routine is to provide an easy method of displaying text; however, since the text is converted to ASCII by using the resident EBCDIC to ASCII conversion table (see the description of EBCASC in MTS Volume 3), textual data can be saved with graphical information on files and copied to the Computek at a later time. (Note: the "mixing" on a file of graphical data and text written by FORTRAN will not work, since FORTRAN writes in EBCDIC.) Text is inserted into the command string by surrounding the desired text with quotes (not primes--primes are used to designate character information in FORTRAN). A quote may be used as text by placing a double quote in the text string. An alternative method of inserting text is described under the utility commands description. Currently, each string that is enclosed between quotes must be less than 255 characters after application of the spacing command; this does not mean that the entire input string to be processed must be less than 255 characters. All blanks between commands and strings are ignored, but other characters will cause an error exit.

CKTXT Commands

Case commands: These commands are used to indicate the type of case conversion to be performed on any text which may follow in the command stream for this particular call to CKTXT. The default setting is CN.

CN - No case conversion desired; all letters are to be given the case they have in the text stream.
CL - Convert all upper-case letters to lower-case; all lower-case letters remain unchanged.
CU - Convert all lower-case letters to upper-case; all upper-case letters remain unchanged.

Vertical/horizontal print commands: These commands are used to indicate whether the following text is to be displayed vertically or horizontally on the screen. These commands also indicate the "meaning" of the line-feed command if it is used. The default setting is LD.

LF - Execute a line feed at this point.
LD - Display text horizontally; if a line-feed command is issued, the next line is to be below the current line.
LU - Display text horizontally; if a line-feed command is issued, the next line is to be above the current line.
LL - Display text vertically; if a line-feed command is issued, the next line is to be to the left of the current line.
LR - Display text vertically; if a line-feed command is issued, the next line is to be at the right of the current line.

Move commands: These are used to move the cursor either visibly or invisibly in the manner specified. Each command may be used in sequential or indexed form. The sequential form employs a moving pointer which "remembers" the last (x,y) pair used and which is updated only after a sequential command is processed. A brief example of what is meant by the sequential and indexed form will help make the difference clear.

Let I represent the sequential pointer (initially, I is preset to zero in a CKTXT call) and let XX be the move command desired; then the legal commands would be:

1) XX - which means CALL CKXX(X(I+1),Y(I+1)), and set I=I+1 after the call.
2) XX...m - which effectively means CALL CKXX(X(I+J),Y(I+J)) with J running from 1 to m, and set I=I+m after the call.
3) XXn - which means CALL CKXX(X(n),Y(n)), and do not change I.
4) XXn...m - which effectively means CALL CKXX(X(J),Y(J)) with J running from n to m and do not change I.

Move commands (cont.)

Note: 1) and 2) above are sequential command forms, while 3) and 4) are indexed command forms; also, 1) and 3) are scalar commands while 2) and 4) are vector commands.

The specific move commands will be described by simply giving the CKLIB routine they call in the scalar form. The vector form of the command (for multiple points) simply calls the same routine n times, where n is the number of points specified.

MA calls CKMA	VA calls CKVA
MI calls CKMI	VI calls CKVI
MS calls CKMS	VS calls CKVS
ML calls CKML	VL calls CKVL

Note: The current implementation only allows vector commands of length 25 or less. If it is desired to have a command which uses a vector of more than 25, it must be broken into two or more commands; for example, VA...45 is illegal, but VA...25VA...20 is legal and will do exactly the same thing that was intended in the VA...45 command.

Utility commands:

#n - a #-sign is used as the command-string terminator and must be present once in every command string. The n following it is a digit which indicates the type of transmission desired.

Sn - the space command is used to indicate the number of spaces that are to appear between characters of the following text. The digit n tells how many spaces desired, where $0 \leq n \leq 9$. The default setting is S0; i.e., no spaces between characters.

B[n] - the base command is used to set the "base" for displaying text. If this command is not used, the current cursor position is acquired and used as the base. The purpose of the base is to indicate where the next line of text should start after a line feed. The line-feed command modifies the base in the following ways:

1) If LD is set, then the base, represented here by (XB,YB), becomes (XB,YB-20).
2) If LU is set, then (XB,YB) becomes (XB,YB+20).
3) If LL is set, then (XB,YB) becomes (XB-12,YB).
4) If LR is set, then (XB,YB) becomes (XB+12,YB).

If n is present (n being the optional digit string), it represents the index into the X and Y vectors passed, where the base setting is to be found; otherwise, the sequential pointer described in the move description will be used to acquire the base setting.

Utility commands: (cont.)

O[n] or O"name" - The object insertion command is used to insert into the output buffer the object specified either by the name in quotes (this name must be four characters long), or, if n (a digit string) is present, by the nth object named in the RGOB vector; otherwise, the object whose name appears next in the RGOB vector will be inserted. A sequential index is kept which "remembers" the last word used in RGOB and is updated anytime an O or R command is given.

R[n] or R"name" - The region activation command is used to activate a new region. The name of the region to be activated is specified in the same manner as that of an object to be inserted.

‹› or ‹n› - This utility command inserts text into the text stream. The purpose of the command is to allow variable text information to be processed. The command can only be used in a text stream (i.e., between a pair of quotes); it takes characters from the string vector passed, STR, and inserts them into the text stream for processing. The form of this command is "...text...d...text...", where ...text... represents a character string of zero or more characters, and d is either ‹› or ‹n›, where n is a digit string that serves as an index into STR. The string vector, STR, must have been passed to use this option, and it must be set up in the manner illustrated by the following example:

Word	Contents and Meaning
STR(1)	1 (hex 00000001), the indicator or "flag" for this parameter in the calling sequence.
STR(2)	5 (hex 00000005), indicating for this example that 5 bytes of text follow.
STR(3)	ABCD, the first four bytes.
STR(4)	E???, the fifth byte and three unknown, irrelevant bytes.
STR(5)	8 (hex 00000008), indicating for this example that 8 bytes of text follow.
STR(6)	ABCD, the first four bytes.
STR(7)	EFGH, the remaining four bytes.

Note: Since ‹ is used as a command delimiter, two of them must be used in the text stream to indicate the literal character ‹. The following command string, CSTR,

'"HI ‹‹‹1›› THERE"#0'

when used in the following call, together with the vector STR, above,

```
CALL CKTXT(CSTR,X,Y,STR)
```

would insert the following character string in the output buffer:

```
HI ⟨ABCDE⟩ THERE
```

Note: Users who have variable character sizes on their Computeks can change the character size in CKTXT by the following command:

CSn - Change character size on output to n ($1 \leq n \leq 4$), if n is not present it is the same as giving the command CS1.

CKTXT will always change the character size back to size one on exit, thus the size must be reset on <u>every</u> call.

Purpose: To perform core-to-core conversion of numbers.

Calling sequence: CALL CKCVT(INTERN,EXTERN,IFORM[,&rc4,...&rc12])

Parameters:

INTERN - a scalar or vector of type integer*4, real*4 or real*8 which contains the internal values on "output" conversion and receives the internal values on "input" conversion.

EXTERN - an integer*4 vector which, on "input" conversion, contains a fullword integer byte-count in word one, followed by a character string of the designated length. On "output" conversion this vector receives a fullword integer byte-count, followed by the character representations of the converted internal values. The converted form is compatible with the variable string option of CKTXT.

IFORM - an integer*4 vector containing the following format information:

Word	Information
1	indicator value for type of conversion desired (see table below).
2	number of items to be converted.
3	the integer 4 if INTERN is integer*4 or real*4; otherwise, the integer 8.
4	on "output" conversion, the width in bytes (characters) of the external field; on "input" conversion the fourth byte of this word is the character to be used as the separator between input items. (Note: a separator must terminate the list of "input" items to be converted.)
5	the number of decimal places to the right of the decimal point.
6	a scale factor, p, such that $0 \leq p \leq 255$, with meaning as follows:

p	INTERN *	p	INTERN *
0	10^0=1	128	10^0=1
1	10^1=10	129	10^{-1}=.1
2	10^2=100	130	10^{-2}=.01
	⋮		⋮
127	10^{127}=big	255	10^{-127}=little

Note: Words 5 and 6 are not required for integer or hexadecimal conversion.

rcn - optional statement label(s) for error return code(s) of n.

Return codes:

0	-	normal return
4	-	unable to perform conversion as specified; conversion stopped (see CKERR, Section 2.3.3)
8	-	character string on "input" conversion was exhausted before the specified number of items were converted
12	-	illegal type specification

Description: CKCVT interfaces the user directly with the FORTRAN I/O package (in particular, with ADCON#), so that core-to-core conversion can be performed. CKCVT pre-processes all "input" conversion so that errors are caught before they become fatal. Pre-processing has led to a few restrictions on what can be "read" by a particular "format"; acceptable forms for "input" conversion of the various number types are:

Type	Form of input
I	optional + or - followed by a digit string
F	optional + or - followed by a digit string which may contain one decimal point.
E	optional + or - followed by a digit string which may contain one decimal point and may also contain an exponent string. If present, the exponent string is an E followed by an optional + or - and a digit string.
D	same as E-type, except that the optional E is an optional D.
Z	a hexadecimal digit string (0-9,A-F).

The major difference from FORTRAN conventions is that embedded blanks are not treated as zeros; if not used as separators (see Word 4 of IFORM, above), they are errors. Output conventions are exactly as FORTRAN's, except that a field overflow results in an error return.

Table of Conversion Types (for Word 1 of IFORM)

Value	Desired Conversion
0	F-type internal to external
1	E-type " "
2	D-type " "
3	I-type " "
4	Z-type " "
5	F-type external to internal
6	E-type " "
7	D-type " "
8	I-type " "
9	Z-type " "

Purpose: To acquire error information about a failure in CKCVT conversion.

Calling sequence: CALL CKCERR(IA1,IA2)

Parameters:

IA1 - a fullword integer which receives either the number of bytes remaining or the number of items remaining, depending on the type of conversion and the type of error.

IA2 - a fullword integer which receives the illegal character found on "input" conversion (left justified and padded with blanks); left unchanged if no "input" error occurred.

Description: CKCERR is used to acquire error information pertaining to the last attempted conversion which failed.

Output conversion

A return code of 4 from CKCVT implies that conversion was unsuccessful because of an inadequate field length. A call to CKCERR will return in IA1 the number of items remaining to be converted, including the one which was not converted. Note: IFORM(2)-IA1+1 is the index of the data item which caused the error. (See CKCVT, Section 2.3.2, for the description of IFORM(2).)

Input conversion

A return code of 4 from CKCVT implies that an illegal character was found in the input stream. A call to CKCERR will return in IA1 the number of bytes remaining in the input string, including the one which was illegal. The illegal character will be returned in IA2.

A return code of 8 from CKCVT implies that the input string was exhausted before the required number of items were found. A call to CKCERR will return in IA1 the number of items not found.

Purpose: To continue or to restart core-to-core "input" conversion.

Calling sequence: CALL CKNEXI(INTERN,IACT,IFORM[,&RCN])

Parameters:

INTERN - a scalar or vector of type integer*4, real*4 or real*8 which receives the internal values on "input" conversion. (This vector may be the same one passed to CKCVT, but need not be.)

IACT - a fullword integer zero or one, 0 indicating that attempted conversion of a "bad" item be restarted, and 1 indicating that conversion continue with the next item.

IFORM - a fullword integer vector containing format information. It is identical in form and effect to the IFORM parameter of a call to CKCVT (see Section 2.3.2). Though this vector may be the same one passed to CKCVT, in general it will not be.

rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return

4 - unable to perform conversion as specified; conversion stopped (see CKCERR, Section 2.3.3).

8 - character string on "input" conversion was exhausted before the requested number of items were converted.

12 - illegal type specification.

Description: CKNEXI is used to "step through" a string of input characters (the same string passed in EXTERN in the previous call to CKCVT) which were not converted as desired by CKCVT. Error returns and CKCERR provide basic information regarding the nature of the conversion problem and CKNEXI provides a method for handling it. In general, those problems take two forms: (a) an improper character within an input data item, in which case the character may be changed or conversion restarted according to different IFORM information with IACT=0; or, (b) an improper character between input data items (e.g., an improper separator), in which case conversion may continue with the next data item, IACT=1. Note: CKNEXI may be called only for "input" conversion and only when CKCVT has been called previously.

Although the generality of CKNEXI requires that the programmer construct his own particular method for resolving input conversion problems, the skeleton of a simple example may clarify one approach that may be taken. Assume that the program expects to find an integer, terminated by a comma, in EXTERN and that the user has typed a letter O instead of a zero. Then CKCVT will make an error return with rc=4. The statement label for rc=4 could be the label for a call to CKCERR, which would provide information on the position of the character in EXTERN and what that character was. After replacing the letter O with a zero, the program could then call CKNEXI with IACT=0 so that another conversion attempt would be made.

Variable size fonts

Purpose: These subroutines provide a method for software generation of characters

Subroutines: CKF1, CKSZF1, CKBLF1, CKITF1, CKXYF1

Calling sequences:
```
CALL CKF1(STR,N[,IB])
CALL CKSZF1(X1[,Y1[,X2[,Y2]]])
CALL CKBLF1(ABL)
CALL CKITF1(AIT)
CALL CKXYF1(STR,N,DX,DY)
```

Parameters:

- STR - a vector containing the EBCDIC characters to be drawn
- N - a fullword integer specifying the number of characters in STR
- X1 - a fullword real specifying the x-direction length of the output character (see figure below).
- Y1 - if present, a fullword real specifying the y-direction length of the output character
- X2 - if present, a fullword real specifying the x-direction length of the cell (see figure below)
- Y2 - if present, a fullword real specifying the y-direction length of the cell
- ABL - a fullword real which specifies the angle (in degrees) that the base line is to be rotated counter-clockwise from the horizontal
- AIT - a fullword real which specifies the angle (in degrees) that the characters are to be italicized
- DX - a fullword real where the x-distance of the final cursor position is to be placed
- DY - a fullword real where the y-distance of the final cursor position is to be placed
- IB - if present, a fullword integer specifying the type of transmission

Default values:

- X1 = 16 (logical units)
- Y1 = 24
- X2 = 24
- Y2 = 40
- ABL = 0 degrees
- AIT = 0 degrees

If Y1, X2 or Y2 are omitted in a call to CKSZF1, they default as follows:

- Y1 = 1.5*X1
- X2 = 1.5*X1
- Y2 = 2.5*X1

Description: CKF1 draws N characters starting at the present cursor position according to the size, baseline and italics parameters currently in effect. These parameters may be changed by calls to the entry points CKSZF1, CKBLF1, and CKITF1. The EBCDIC characters LF (line feed) and BS (backspace) are accepted and treated appropriately. The characters ! and % are interpreted as LF and BS respectively so that literals may be used for STRING. The cursor is left at the lower left hand corner of the cell after the last character. Thus, consecutive calls may be made to CKF1. CKXYF1 does not draw the characters; rather, it computes the logical distances the cursor will travel if CKF1 is called on the same STRING.

CKCSIZ

Purpose: To change the character size on a Computek terminal equipped with multiple character sizes

Calling sequence: CALL CKCSIZ(IC[,ICP])

Parameters:

IC - a fullword integer used to set the character size
- 0 - return present size
- 1 - regular character size
- 2 - double size characters
- 3 - triple size characters
- 4 - quadruple size characters

ICP - if present and if IC=0, then this argument contains an integer (1, 2, 3, or 4) designating the current character size.

Description: This routine places the appropriate information into the output buffer to set the character size as specified in IC, or it returns the present character size setting in ICP if IC is 0.

Curve and Vector Fitting

Many graphical figures are designed to represent a "smoothly curving" line. Obvious examples include the graphical representation of continuous functions and the pictorial silhouette of (most) human anatomies. Data for the first example are generally computed for certain key, or regularly incremented, values, and the resulting coordinates are used as the basis for "filling-in" the contours. We will confine our attention to problems of that sort; for the second example, we will leave the determination of key values, data collection procedures and filling-in techniques to the imagination of the reader.

These routines are designed to assist in the graphical reproduction of smooth contours for which a limited number of coordinate values are known. CKVFIT inserts intermediate points, producing a smoother line, in general, than would CKVABS if called with the same coordinate vectors. (It should be noted that CKVFIT, unlike CKVABS, uses the first point passed--not the current cursor position--as the starting point for the first vector.) CKSMTH also adds intermediate points, but uses a different procedure for determining where points should be inserted; unlike CKVFIT, the enlarged set of points is not automatically inserted into the output buffer but is returned for use by the calling program. CKCFIT, on the other hand, reduces the number of output points, if possible, by testing whether a single curve can connect more than two of the given points without exceeding the specified error tolerance.

It would be difficult to establish which of the available methods would provide the "best" contour for any reasonably general class of point-vectors; there is a definite trade-off between accuracy and economy. However, given a particular contour, the programmer should be able, by trying various combinations of routines, to reproduce the contour efficiently and accurately.

CKVFIT

Purpose: To fit a "nearly uniform" set of straight lines to a set of points.

Calling sequence: CALL CKVFIT(X,Y,NPT,IB)

Parameters:

- X - a real*4 vector containing the x-coordinate of the logical addresses.
- Y - a real*4 vector containing the y-coordinate of the logical addresses.
- NPT - a fullword integer specifying the number of points.
- IB - a fullword integer specifying the type of transmission. This parameter is <u>not</u> optional.

Description: CKVFIT is used to connect a set of points by a nearly uniform set of straight lines. A "nearly uniform" set of straight lines means that the points passed are connected directly if the sum of the x and y components of the distances between the two points does not exceed 75 units; otherwise, intermediate points are inserted at equal intervals along the x-axis. The net result of this process is to give a reasonably good straight-line approximation to graphical pictures containing arcs.

Purpose: To fit a minimal sequence of curves to a set of points.

Calling sequence: CALL CKCFIT(X,Y,NPT,DIS,LEN,TOL,IB)

Parameters:

- X - a real*4 vector containing the x-coordinate of the logical addresses.
- Y - a real*4 vector containing the y-coordinate of the logical addresses.
- NPT - a fullword integer indicating the number of points.
- DIS - a work vector whose size should be at least 4*NPT.
- LEN - the length of the vector DIS.
- TOL - a real*4 tolerance to be used in fitting. If TOL $\leq$ 0, 4 logical units will be used; otherwise, TOL logical units will be used.
- IB - a fullword integer specifying the type of transmission. This parameter is not optional.

Description: CKCFIT fits a minimal sequence of curves to a set of points. A least squares fit is used, fitting the points by means of the curvature equations to within tolerance specified. Note: CKCFIT sometimes eliminates too many points and the picture generated may not be acceptable. If this occurs, there are several things to try:

1) decrease the tolerance (the closer to one logical unit the more demanding the fit);
2) use the routine CKSMTH; or,
3) add intermediate points.

Purpose: To add points to a set of connected points at any place where a change in the sign of the slope occurs.

Calling sequence: CALL CKSMTH(XV,YV,NPT,XXV,YYV,LEN)

Parameters:

XV - a real*4 vector containing the x-coordinate of the logical addresses.
YV - a real*4 vector containing the y-coordinate of the logical addresses.
NPT - a fullword integer indicating the number of points.
XXV - a real*4 vector used for the generated x-coordinates of the logical addresses.
YYV - a real*4 vector used for the generated y-coordinates of the logical addresses.
LEN - a fullword integer where the length of the generated vector will be stored.

Note: The dimension of XXV and YYV should be at least NPT+6*(the number of sign changes in the slope); that dimension need not be more than 4*NPT-3.

Description: CKSMTH examines the slopes of the lines connecting each 3-tuple of points. If the signs of these two slopes are different, then it adds 3 points between the first point and the second point and 3 points between the second and third points. The x and y vectors passed are not altered; rather, new x and y vectors are generated (XXV and YYV respectively). The length of these new vectors, i.e., the new number of points, is stored in LEN; thus, LEN-NPT equals the number of points added by CKSMTH.

Purpose: To prepare for display a labelled linear graph and optionally display a graph.

Calling sequence:

CALL CKGRAF(DOPE,NPTS,X,Y,XCH,NXCH,YCH,NYCH,TCH,NTCH,PCH[,XSZ,YSZ])

Parameters:

DOPE - a real*4 vector of length 12 which provides positioning and scaling information; only the first 6 elements are specified by the user.
- DOPE(1) - x-coordinate of the lower left corner of the grid
- DOPE(2) - y-coordinate of the lower left corner of the grid
- DOPE(3) - x-coordinate of the upper right corner of the grid
- DOPE(4) - y-coordinate of the upper right corner of the grid
- DOPE(5) - x-directional grid increment
- DOPE(6) - y-directional grid increment
- DOPE(7) - x scale factor
- DOPE(8) - y scale factor
- DOPE(9) - minimum x grid value
- DOPE(10)- minimum y grid value
- DOPE(11)- maximum x grid value
- DOPE(12)- maximum y grid value

NPTS - a fullword integer specifying the number of points to be plotted. If NPTS is negative a solid line will be drawn between all points

X - a real*4 vector of x-coordinates to be plotted

Y - a real*4 vector of y-coordinates to be plotted

XCH - a vector of characters to be used in labelling the x-axis

NXCH - a fullword integer specifying the number of characters in XCH. If NXCH is negative only tick marks are drawn.

YCH - a vector of characters to be used in labelling the y-axis

NYCH - a fullword integer specifying the number of characters in YCH. If NYCH is negative only tick marks are drawn.

TCH - a vector of characters to be used in titling the graph

NTCH - a fullword integer specifying the number of characters in TCH

PCH - a 3-character string: the first character is the plotting character, the second two characters are digits specifying that every nth point is to be plotted, i.e., X02 means that X is the plotting character and is to appear on every second point.

XSZ - if present, a fullword integer which specifies the character size to be used for the numbers which appear at the x grid marks. Note, this option is useful only if the terminal is equipped with variable character sizes.
Default - 1
1 - regular character size
2 - double size
3 - triple size
4 - quadruple size

YSZ - if present XSZ must be present, a fullword integer which specifies the character size to be used for the numbers which appear at the y grid marks.
Default - 1

Description: The user specifies where he wants to position his graph on the Computek screen ($0 \leq x \leq 1023$, $0 \leq y \leq 800$) through the first four elements of the DOPE parameter. Note that when standard size characters are used a strip of about 125 raster units in width must be allowed for at the bottom and left of the graph. These regions are below and to the left of the coordinates given in DOPE(1) and DOPE(2). If he is not sure of the range of data to be plotted he provides guesses for the grid increments via DOPE(5) and DOPE(6). Furthermore, values of DOPE(7),..., DOPE(12) are returned for possible utilization by the user. When the user is sure of the range of his data he can enter CKGRAF with NPTS=2 and lay out the graph exactly as he wants it. In this case X(1), Y(1), and X(2), Y(2) are the values of x and y to be assigned to the lower left and upper right corners of the graph, respectively. Subsequent plotting would be done via CKPPLT. The grid is not transmitted but left in the buffer.

CKPPLT

Purpose: To display a graph on a grid prepared by CKGRAF

Calling sequence: CALL CKPPLT(DOPE,NPTS,X,Y,PCH)

Parameters:

DOPE - a real*4 vector of length 12 which provides positioning and scaling information; only the first 6 elements are specified by the user.

NPTS - a fullword integer specifying the number of points to be plotted. If NPTS is negative a solid line will be drawn between all points.

X - a real*4 vector of x-coordinates to be plotted.

Y - a real*4 vector of y-coordinates to be plotted.

PCH - a 3-character string: the first character is the plotting character, the second two characters are digits specifying that every nth point is to be plotted, i.e., X02 means that X is the plotting character and is to appear on every second point.

Description: CKPPLT cannot be called unless a prior call was made to CKGRAF. CKPPLT is used to plot a graph composed of NPTS points on the grid prepared by CKGRAF.

Special Figures

Routines in this group are a step removed from those in the basic graphics group. Each of them requires the minimum amount of information needed to produce standard figures frequently used in display construction. Parameters supplied in the call are used to calculate specific values which then are used in multiple calls to the basic routines. The function and usefulness of CKRECT and CKDASH are obvious from their descriptions; the "circle" and "arc" routines provide reasonably accurate and easily specified approximations to figures which are heavily used but are not immediately compatible with the terminal's built-in vector and curve capabilities.

As indicated in the Introduction (1.10), each programmer will have his own idea of what constitutes a "special figure", and there are several ways by which he can develop his own "library" of them. Although this group of CKLIB routines quite possibly will grow if a demand develops for particular additional figures, only those of general applicability will be considered. Further information may be found in Appendix D.

PAGE 2.5.1 - CKCIR - MISSING FROM DOCUMENT PRIOR TO
ITS BEING SHIPPED TO EDRS FOR FILMING

BEST COPY AVAILABLE

CKARCS

Purpose: To draw a circular arc with n straight-line segments.

Calling sequence: CALL CKARCS(RAD,ANG1,ANG2,NS,IB)

Parameters:

RAD - the real*4 radius of the circle in logical address units.
ANG1 - the real*4 angle in degrees at which the arc is to begin.
ANG2 - the real*4 angle in degrees at which the arc is to end.
$0 \leq ANG1 \leq ANG2 \leq 360$
NS - a fullword integer specifying the number of straight line segments to use.
IB - a fullword integer specifying the type of transmission. This parameter is not optional.

Description: CKARCS draws any portion of a circle centered at the current cursor position by using NS straight line segments. The cursor is returned to the center of the arc before return to the calling program.

Purpose: To draw a circular arc with curved-line segments.

Calling sequence: CALL CKARC(RAD,ANG1,ANG2,IB)

Parameters:

RAD - the real*4 radius of the circle in logical address units.
ANG1 - the real*4 angle in degrees at which the arc is to begin.
ANG2 - the real*4 angle in degrees at which the arc is to end.
$0 \leq ANG1 \leq ANG2 \leq 360$
IB - a fullword integer specifying the type of transmission. This parameter is not optional.

Description: CKARC draws smooth curves for any portion of a circle centered at the current cursor position. The cursor is returned to the center of the arc before return to the calling program.

Purpose: To draw a rectangle.

Calling sequence: CALL CKRECT(H,V,IB)

Parameters:

H - the real*4 length of the horizontal side in logical address units.
V - the real*4 length of the vertical side in logical address units.
IB - a fullword integer specifying the type of transmission. This parameter is not optional.

Description: CKRECT draws a rectangle with the lower left corner at the present cursor position. The horizontal side is of length H and the vertical side of length V. The cursor is left at its original position upon return to the calling program.

Purpose: To draw a dashed line between two points.

Calling sequence: CALL CKDASH(X1,Y1,X2,Y2[,LEN][,IB][,&rc4])

Parameters:

X1 - the x-coordinate of the logical address of the starting point
Y1 - the y-coordinate of the logical address of the starting point
X2 - the x-coordinate of the logical address of the ending point
Y2 - the y-coordinate of the logical address of the ending point
LEN - if present, the length in logical address units of the dash to be used
IB - if present, a fullword integer specifying the type of transmission
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - negative length specified

Description: CKDASH is used to draw a uniform dashed line between two points. The lengths of the intermediate dashes and of the spaces between dashes are equal and may be specified. The dash length is computed, if not specified, by the following algorithm:

```
L=(X1-X2)**2+(Y1-Y2)**2
LEN=20
IF (L.LT.100)LEN=L/5
```

Region Management

The reader may find it helpful to review material in the Introduction (1.7ff), the Note in Section 2.1 and the third paragraph of Section 2.2.0. He is also encouraged to concentrate first on understanding the geometrical effects of changing the logical region (i.e., in effect, "renaming" the grid points); these concepts are not difficult and, once grasped, seem trivial. However, the algebra and arithmetic required to implement these concepts, though straightforward, are frequently complex and always laborious; that is why this group of routines was written.

When a new logical region is defined, via CKREG, the origin is "moved" from the lower-left corner of the screen (translating the x and y axes), then the new coordinates are scaled (distances are "stretched" or "squeezed") so that the "contents" of the original 1024x1024 square grid now occupy the rectangle specified in the subroutine call. (It should be noted that the new rectangle may lie entirely within, partially within or entirely outside the original square: it is perfectly legal to specify region boundaries outside the 0-1023 ranges; the effects of "wraparound," a physical feature of the terminal, do not show up until actual cursor movements are computed.) The new rectangle may then be translated--retaining its scale factors--by moving its origin via a call to CKORG; or, it may be re-scaled--its origin remaining fixed--via CKSCA. Although it is possible to change the origin and/or scale factors of the base region, it is suggested that, instead, a new region be defined with 0 and 1023 boundaries and then translated or scaled appropriately.

A CKLIB graphical routine has performed its primary function when graphic commands (cursor movements) have been inserted into the output buffer. These commands consist largely of coordinate information--either addresses or increments--which specifies integral points in the terminal's hardware address grid. However, that coordinate information generally will bear little resemblance to the values passed to the routine originally. Logical addresses, by which a program specifies cursor movements in the currently active region, are automatically transformed into physical addresses as follows:

Physical address = (region origin) + (scale factor) * (logical address),

where "scale factor," or overall scale, is the product of the region scale (from CKREG) and the user scale (from CKSCA). Logical increments, of course, are simply multiplied by the scale factor. We can abbreviate the above formula and represent its x-coordinate and y-coordinate components as:

$$PA_x = ORG_x + RS_x * US_x * LA_x$$

$$PA_y = ORG_y + RS_y * US_y * LA_y$$

When the predefined base region is the active region, default values are used and the transformation becomes:

$$PA_x = 0 + 1*1*LA_x = LA_x$$

and similarly for the y-coordinate. As implied above, a call to CKREG will, in general, change the values of ORG and RS, while calls to CKORG and CKSCA will alter ORG and US, respectively.

Before we pass to a concrete example of region management, a few general remarks should help clarify the role of the region routines with respect to the rest of CKLIB. There is always an active region whose coordinate system is used to transform logical cursor movements into physical movements; since those transformations occur within the CKLIB routines, the programmer must ensure that the logical addresses (and increments) passed to the routines are appropriate for the region which is active at that time. The corresponding condition holds also for CKRC and CKRCRT (Section 2.10). It should be noted, in that context, that calls to CKTXT (2.3.1) may dynamically change the region and that calls to CKVABW (2.7.1) return after the re-activation of the base region.

Figure 4 illustrates successive transformations in the logical addresses of selected screen positions (physical addresses). Those transformations would accompany the sequence of three calls indicated in the figure. For clarity, logical addresses have been rounded off. The net effect of these transformations would be to map logical addresses in the range ($1.0 \le x \le 3.0$, $5.0 \le y \le 10$) onto the shaded portion of the screen. The reader would do well to refer to the above formulas and the respective routine descriptions in tracing the sequence of transformations, perhaps most easily in terms of the changing "origin." He may also wish to verify, or use, the fact that the region bounded by logical addresses

(xmin, xmax, ymin, ymax)

may be mapped onto the screen area bounded by physical addresses

(XMIN, XMAX, YMIN, YMAX)

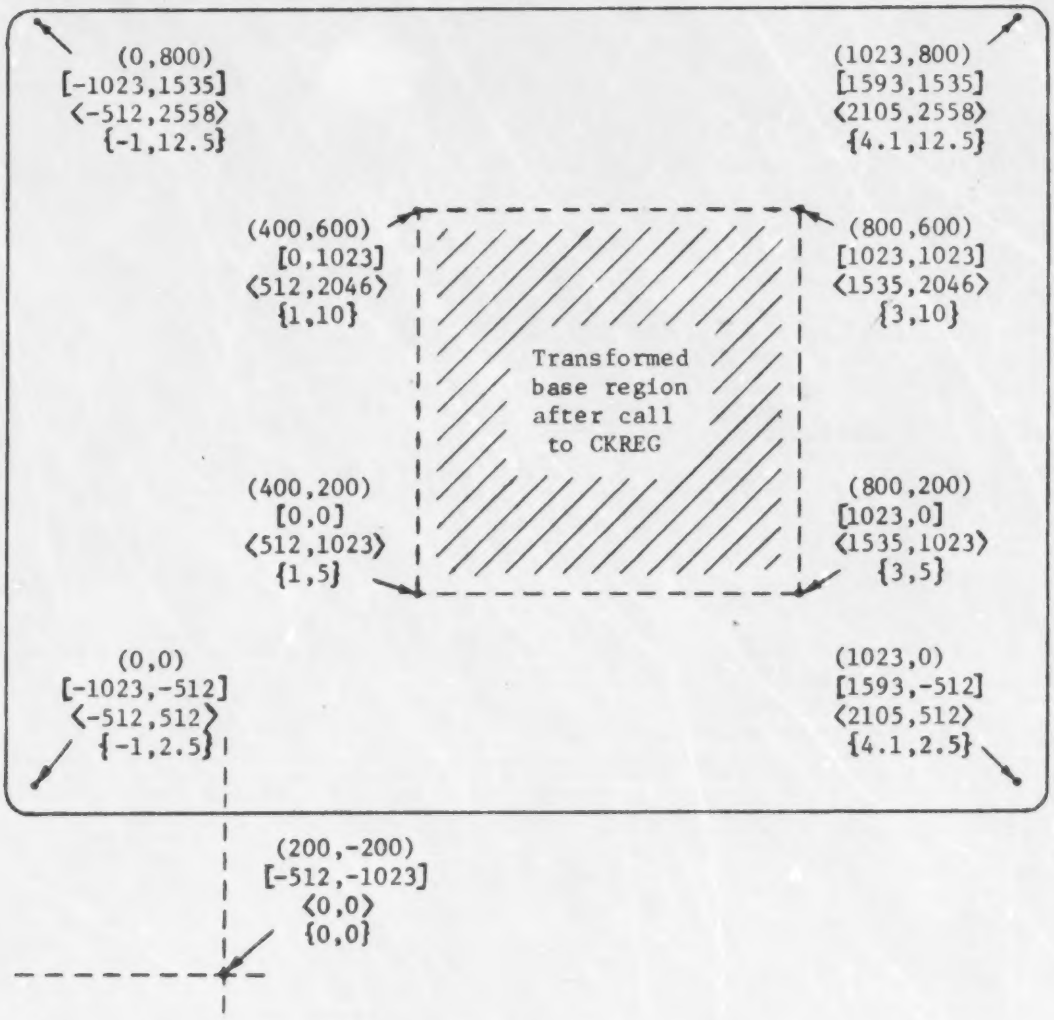
by setting:

$$US_x = \frac{XMAX-XMIN}{xmax-xmin} ;$$

$$US_y = \frac{YMAX-YMIN}{ymax-ymin} ;$$

$$ORG_x = XMAX - US_x*xmax \text{ ; and,}$$

$$ORG_y = YMAX - US_y*ymax .$$



Coordinate Transformation	Notation
(Predefined base region)	(x,y)
CALL CKREG(REGN,400.,800.,200.,600.)	[x,y]
CALL CKORG(REGN,200.,-200.)	⟨x,y⟩
CALL CKSCA(REGN,1023./2,1023./5)	{x,y}

Figure 4

The transformation illustrated in Figure 4, then, could also have been accomplished by

```
CALL CKREG(REGN,0.,1023.,0.,1023.)
CALL CKSCA(REGN,200.,80.)
CALL CKORG(REGN,200.,-200.)
```

Example 4 in Section 3 provides another illustration of region management--one in which the final display exhibits the flexibility of the routines in significantly reducing the labor of planning and programming multiple figures of similar but variable design.

Purpose: To define by name, and establish initial values for, a new logical region.

Calling sequence: CALL CKREG(REGN,XMIN,XMAX,YMIN,YMAX[,&rc4])

Parameters:

REGN - a fullword whose contents is to be the name for the region. Four character zeros (hex F0F0F0F0) is the name of the predefined base region.
XMIN - the real*4 minimum bound in the x-direction
XMAX - the real*4 maximum bound in the x-direction
YMIN - the real*4 minimum bound in the y-direction
YMAX - the real*4 maximum bound in the y-direction
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.
Note: The four boundary values are expressed in logical addresses of the predefined base region.

Return codes:

0 - normal return
4 - XMIN greater than XMAX or YMIN greater than YMAX.

Description: CKREG is used to define a logical region. It sets the upper vertical bound to YMAX, the lower vertical bound to YMIN, the left-most horizontal bound to XMIN and the right-most horizontal bound to XMAX. In addition, it initializes the origin of the region to (XMIN,YMIN) and the region scale factors to:

$$RS_x = \frac{XMAX-XMIN}{1023}$$

$$RS_y = \frac{YMAX-YMIN}{1023}$$

The newly defined logical region then becomes the active region.

Purpose: To change the origin of a logical region and make it the active region.

Calling sequence: CALL CKORG(REGN,X,Y[,&rc4])

Parameters:

REGN - a fullword whose contents is the name of a region
X - the real*4 x-coordinate of the new origin
Y - the real*4 y-coordinate of the new origin
rcn - optional statement label(s) for transfer on non-zero code(s) of n.
Note: X and Y are expressed in logical addresses of the predefined base region.

Return codes:

0 - normal return
4 - region name specified has not been defined

Description: CKORG is used to change the origin of the specified logical region from its default value--the point (XMIN,YMIN) of the call to CKREG--to the specified origin (X,Y). This routine may also be used to change a previously set origin. The region specified becomes the active region.

Purpose: To change the scale factors of a logical region and make it the active region.

Calling sequence: CALL CKSCA(REGN,XSCALE,YSCALE[,&rc4])

Parameters:

REGN	-	a fullword whose contents is the name of a region
XSCALE	-	the real*4 scale factor for the x-direction
YSCALE	-	the real*4 scale factor for the y-direction
rcn	-	optional statement label(s) for transfer on non-zero code(s) of n.

Return codes:

0	-	normal return
4	-	region name specified has not been defined

Description: CKSCA is used to reset scale factors for the x-direction and the y-direction in the specified region. The result of setting scale factors is that all logical addresses and relative distances processed while the named region is active will have their x and y components multiplied by XSCALE and YSCALE, respectively. (See Section 2.6.) The region specified in the call becomes the active region.

Purpose: To retrieve the description of the active region.

Calling sequence: CALL CKGRIN(VEC)

Parameters:

VEC - a real*4 vector of length 8 where the region description is to be copied.

Description: CKGRIN provides a method for obtaining the active region description. It returns the following information in the vector passed:

Word	Contents
1	minimum x boundary
2	maximum x boundary
3	minimum y boundary
4	maximum y boundary
5	x-coordinate of the origin
6	y-coordinate of the origin
7	scale factor for the x-direction
8	scale factor for the y-direction

Values returned will be those specified for the region by the most recent calls to CKREG, CKORG or CKSCA, except that the returned scale factors will be overall scale factors (see Section 2.6). Note: changing a value in the vector does not change that value in the internal region description.

Purpose: To make a previously defined logical region the active region.

Calling sequence: CALL CKACT(REGN[,&rc4])

Parameters:

REGN - a fullword whose contents is the name of a logical region.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - region name specified has not been defined

Description: CKACT causes the region named in the call to be made the active region if it exists. If the region named does not exist, an error return is given and the currently active region remains active.

Purpose: To inactivate the active region.

Calling sequence: CALL CKIACT

Description: CKIACT is used to inactivate the current active region, and reactivate the base region. Since the base region may have been altered by re-setting its scale factors and/or its origin, it is reset to initial conditions by this routine before being made active. This means that the x and y scale factors are set to one and the origin is set to the physical address (0,0).

Purpose: To destroy the specified logical region(s).

Calling sequence: CALL CKDREG(REGN1[,REGN2,...,REGNn])

Parameters:

REGNn - one or more fullwords which contain region names

Description: CKDREG is used to destroy the region(s) specified in the call. (All regions require storage space to contain the region description. This routine provides a means of de-allocating that space when it is no longer needed. It also removes the region name from the list of available regions, thus reducing the amount of search time required when a region name is sought.) The base region cannot be destroyed, but an attempt to do so will reset its origin and scale factors to their pre-defined, default values.

Purpose: To provide a method for converting from logical to physical coordinates

Calling sequence: CALL CKPA(XL,YL,XP,YP[,REGN][,&rc4])

Parameters:

XL - a real*4 variable containing the x coordinate of a logical address
YL - a real*4 variable containing the y coordinate of a logical address
XP - a real*4 variable containing the x coordinate of a physical address
YP - a real*4 variable containing the y coordinate of a physical address
REGN - an optional fullword variable containing the region with respect to which coordinate conversion is to take place. If omitted the current active region is assumed.

Return codes:

0 - normal return
4 - region name specified has not been defined

Description: CKPA is used to retrieve physical coordinates of a point defined with respect to a region (not necessarily the active region). There also exist two functions which are entry points to CKPA; they are CKPAX and CKPAY. The calling sequences of these functions are:

```
XP = CKPAX(XL[,REGN])
YP = CKPAY(YL[,REGN])
```

where XP,YP,XL,YL and REGN are defined as above. Note, the active region is not altered by use of these routines, no matter what is specified by the REGN parameter.

Purpose: To provide a method for converting from physical to logical coordinates

Calling sequence: CALL CKLA(XP,YP,XL,YL[,REGN][,&rc4])

Parameters:

XP - a real*4 variable containing the x coordinate of a physical address
YP - a real*4 variable containing the y coordinate of a physical address
XL - a real*4 variable containing the x coordinate of a logical address
YL - a real*4 variable containing the y coordinate of a logical address
REGN - an optional fullword variable containing the region with respect to which coordinate conversion is to take place. If omitted the current active region is assumed.

Return codes:

0 - normal return
4 - region name specified has not been defined

Description: CKLA is used to retrieve logical coordinates of a point defined with respect to a region (not necessarily the active region). There also exist two functions which are entry points to CKLA; they are CKLAX and CKLAY. The calling sequences of these functions are:

```
XL = CKLAX(XP[,REGN])
YL = CKLAY(YP[,REGN])
```

where XL, YL, XP, YP and REGN are defined as above. Note, the active region is not altered by use of these routines, no matter what is specified by the REGN parameter.

Windowing

The visual effect of CKVABW is implicit in the term "window"; some particular area of a planned display is produced, or "drawn", while the remainder of the display is not drawn, "lost", beyond the boundaries of the window. The use of CKVABW is straightforward; but, since it operates on logical regions, the programmer must have a basic familiarity with CKLIB region management.

In addition to the base region, which is made the active region just before CKVABW returns to the calling program, at most two other logical regions are involved. When CKVABW is called, the current active region--call it the display region--provides the coordinate system used in transforming the given logical addresses into physical addresses (screen positions). A second region--call it the window region--is the region named in the calling sequence. That region determines whether each of the cursor movements specified in the call (let's assume they were all specified as visible moves) should actually take place on the screen; i.e., whether the graphic commands for those movements should be placed in the output buffer. The boundaries established in the creation of the window region (via CKREG) affect the cursor movement in one of four ways:

(1) if both endpoints of the move fall within the boundaries, the entire vector is drawn;
(2) if one endpoint falls within the boundaries, only that part of the vector which lies within the boundaries is drawn;
(3) if neither endpoint falls within the boundaries, then either no vector is drawn; or,
(4) only that part of the vector which passes through the bounded area is drawn.

The foregoing description is just a more precise way of saying that the user will see only that part of the display region which falls within the boundaries of the window region.

One particularly convenient application of windowing--control of wraparound--may involve only the predefined base region (see 2.6.1). If the base region is also the active region, then calling CKVABW and naming the base region also as the window region will suppress wraparound and truncate vectors at the boundaries of the physical address grid. Many other uses of windowing are possible, of course, adding considerable visual appeal to a wide variety of applications.

Purpose: To provide a method for moving the cursor n times within a specified region.

Calling sequence: CALL CKVABW(X,Y,NPT,IBEAM,REGN,IT)

Parameters:

- X - a real*4 vector containing the x-coordinates of the logical addresses.
- Y - a real*4 vector containing the y-coordinates of the logical addresses.
- NPT - a fullword integer specifying the number of points.
- IBEAM - a fullword integer vector indicating whether to move visibly or invisibly.
- REGN - a fullword containing the name of the region to be used.
- IT - a fullword integer specifying the type of transmission. This parameter is not optional.

Description: CKVABW is used to move the cursor NPT-1 times entirely within a region. The region named becomes a "window"; all vectors which normally would go outside the boundaries of the window region are truncated at those boundaries. (The move to the first point from the previous cursor position is always invisible, unlike CKVABS.) Thus, the net effect of this routine is similar to that of looking through a window at a display, the boundaries of the window region being the boundaries of the display. Note: the active region after a call to CKVABW is the base region.

Buffer Management

The role of CKLIB's output buffer is described in the section on Display Management (1.11ff); conventions regarding the transmission parameter are presented in the section on Mnemonics, Conventions and Notation (2.1ff). Familiarity with that information is assumed for the following discussion.

These routines were designed primarily to provide alternatives to the handling of buffer contents which occurs under default conditions. While CKTRAN, CKTRH and CKPURG perform functions equivalent to those of the transmission parameters--transmitting and emptying the buffer--the remaining routines are concerned with the directing of buffer transmission to alternative locations. This re-direction of buffer output may be "temporary" or "permanent".

The contents of the output buffer may be transmitted--transferred--to a line file, by CKSAVE, or to the core location named in a call to CKCOB. (For a description of CBCOB and other "object" routines, see 2.9ff.) In both cases, the location receiving the output must be specified for each transmission. (When used together with CKRSTO or the MTS COPY-command, CKSAVE provides a convenient means for "saving" a display and reproducing it without having to repeat the computation required for its initial production.) These two methods, though "temporary" in the sense that they apply only to a single transmission, will always cause output to be transferred to the specified location; they override the effects of "permanent" output assignment as described in the next paragraph.

If logical I/O unit SPUNCH is not assigned in the RUN command, CKLIB will make the initial assumption that its output is to be sent to *MSINK*, which will continue to receive that output until a specific reassignment is made. "Permanent" reassignment of the current output location--that is, the place where the buffer's contents are sent unless temporarily overridden--is accomplished via a call to CKOPUT; the location specified in the call will then receive CKLIB transmission until a different location is specified via another call to CKOPUT.

Four programming details bear mention at this point, since they may otherwise be missed: (a) the output buffer is not purged after a call to CKTRH or a transmission parameter of 2--if the buffer is not then emptied by CKPURG, the same contents will be transferred again the next time the buffer is transmitted; (b) the first transmission after a call to CKCOB does not produce an immediate display--the buffer contents, which thereby

become the value of the object created, must be retained in the buffer or re-inserted into the buffer to produce a visible display; (c) a call to CKRSTO will not affect the output buffer; and (d) the sequence:

```
CALL CKBLEN(LEN)
CALL CKOPUT(CORE)
CALL CKTRAN
CALL CKOPUT('*MSINK* ')
```

will transmit the buffer's contents to a core vector, where it may be "massaged" as the programmer desires. If the total length of those contents is not changed, then CALL CKSB(CORE,LEN) will restore them to the current end of the buffer. Note that '*MSINK* ' in the above call could be replaced by any valid output location.

Purpose: To transmit the contents of the output buffer.

Calling sequence: CALL CKTRAN

Description: A call to CKTRAN causes the transmission of everything which has been placed into the output buffer since the last time it was initialized. After transmission has occurred, the buffer is re-initialized.

Purpose: To transmit the contents of the output buffer without re-initialization.

Calling sequence: CALL CKTRH

Description: A call to CKTRH causes the contents of the output buffer to be transmitted to the previously specified location. After the transmission is completed, control is returned to the calling program and the information contained in the output buffer remains intact, thus allowing successive transmissions of the same information without re-generating it.

CKPURG

Purpose: To re-initialize the output buffer.

Calling sequence: CALL CKPURG

Description: CKPURG is used to re-initialize the output buffer so that the next insertion of information to be displayed will be placed at the beginning of the buffer; i.e., the output buffer is "purged" of all its contents.

Purpose: To obtain the current length of the output buffer.

Calling sequence: CALL CKBLEN(LEN)

Parameters:

LEN - a fullword integer which, upon return, will contain the length (in bytes) of the buffer. Note: this length is the overall length and not the transmission length.

Description: CKBLEN gives the user access to the length of his current output buffer. The transmission of this buffer occurs in 104-byte blocks; thus, the length returned is not the transmission length, unless it is less than 104 bytes. The number of output records required to transmit the entire buffer can be found in the following fashion:

```
CALL CKBLEN(LEN)
IREC = LEN/104
IF (IREC*104 .LT. LEN) IREC=IREC+1
```

Purpose: To have the contents of the output buffer written to a line file.

Calling sequence: CALL CKSAVE(LDN,IREC[,IB][,&rcn])

Parameters:

- LDN - a fullword integer between 0 and 19 inclusive, specifying which logical device unit to use in writing the buffer.
- IREC - a fullword integer specifying the line number at which to begin writing.
- IB - if present, is a fullword integer 1 or 2 indicating whether the buffer is to be purged or not after transmission. The default is one, purge.
- rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

- 0 - normal return
- 4 - device or file unavailable, or illegal logical device number.
- 8 - unable to write record.

Description: CKSAVE writes the contents of the output buffer into the line file attached to unit LDN, starting at the line number specified and incrementing by .001 until the buffer has been completely written onto the file. The choice of .001 as the increment allows displays of less than 100,000 bytes to be saved before the next integral line number is reached, thus allowing IREC to be a record pointer. After the output buffer has been written to the file, this routine adds a $CONTINUE WITH *MSOURCE* RETURN command so that the file may be copied to the Computek with a pause occurring at the end of each logical record. (An end-of-file--ctrl-c--entered from the terminal would then cause the next logical record to be copied.) Note: Information in the file is in binary form; the file is copied to the Computek in the following manner:

```
$COPY fname *MSINK*@BIN
```

The @BIN modifier forces the contents of the file to be treated as binary; thus, information written into this file by other means must first be converted to binary form. Individual logical records may also be displayed via CKRSTO.

Purpose: To restore information (which was written in a file by the routine CKSAVE) to the terminal.

Calling sequence: CALL CKRSTO(LDN,IREC[,&rcn])

Parameters:

- LDN - a fullword integer between 0 and 19 inclusive, specifying which logical device unit to read.
- IREC - a fullword integer specifying the line number at which to begin reading.
- rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

- 0 - normal return
- 4 - device or file unavailable, or illegal logical device number

Description: CKRSTO will copy logical records out of a line file and send them to the current output location. It is important that the file attached to the specified logical device unit be one that was constructed by CKSAVE.

Purpose: To designate the location where the output buffer should be transmitted.

Calling sequence: CALL CKOPUT(LDN | FDNAME | CORE[,&rcn])

Parameters:

LDN	-	a fullword integer logical device unit 0 to 19 where the output is to be written.
FDNAME	-	a file or device name, followed by one or more blanks, designating where output is to be written.
CORE	-	a storage array of some kind where output is to be written.
rcn	-	optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0	-	normal return
4	-	attempt to change sink to core while in object definition mode
8	-	device or file unavailable, or illegal logical device number

Description: CKOPUT is used to change the output location. It cannot be used to change the output location to core while an object is being created, nor can the output location be core while an object is being created. It should be noted that a user who sets the output location to core must be sure that he provides enough space for the largest output buffer that is intended to be transferred to that location. The output routines will begin writing at the specified core location on every request to transmit.

If a file is designated, it must have existed before being referenced by CKOPUT, though it need not have been attached to a logical I/O unit if it is referenced by name.

Object Definition and Use

The "object" routines provide the programmer with a facility comparable to that provided by the "region" routines. In each case a name may be associated with a particular graphical feature which can be referenced thereafter by name only. Each logical region provides its own coordinate system for interpretation of logical addresses; the complex computations required for address translation are performed "transparently" by CKLIB whenever the region is activated. A graphic object, on the other hand, consists of a sequence of graphic commands which may be manipulated as a logical unit and displayed repeatedly, combined as a partial display with other graphic commands or objects and stored for later use in a graphic object file. Again, the "details" required for object manipulation are performed by internal features of the CKLIB routines.

If the programmer has formed a clear conception of CKLIB's output buffer and buffer management, the definition and use of graphic objects will be straightforward. A call to CKCOB effectively declares the buffer "reserved", temporarily, for object creation. When the next buffer transmission occurs--whether by CKTRAN,CKTRH or a transmission parameter--the current contents of the buffer are automatically transferred to a core location which CKLIB has prepared for it under the name specified in the CKCOB call. (The buffer is then purged or not, depending on the type of transmission requested.) The output string just transferred has now become a graphic object, whose name was specified in the call to CKCOB and whose value is the transferred output string.

Now that a graphic object has been defined, it is important to distinguish between its core (memory) version and its file version. An object, once created, may be saved (though it need not be) in a graphic object file which has been opened and initialized by CKOFIL. "Saving" is done by CKSOB, which copies the core version of the object into an equivalent file version; the file version is automatically catalogued internally under the same name. It is the core version which is created (CKCOB), saved (CKSOB), destroyed (CKDOB) and inserted into the output buffer (CKIOB), while it is the file version which is gotten (CKGOB) and whose name is listed with the names of other objects in the file (CKLOB).

The general description of buffer-management routines (2.8.0) contains further information on object handling, as do the CKTXT writeup (2.3.1ff) and Examples 4 and 6 in Section 3. Two additional points should be stressed to prevent confusion and facilitate object handling. A graphic object file is designed for access via CKGOB and cannot be copied directly or selectively, as can a file written by CKSAVE (2.8.5). Since a graphic object is composed of graphic commands in their final, output form, an object composed entirely of incremental cursor movements may be

displayed at various screen locations by re-positioning the cursor before inserting the object into the output buffer (again, see Examples 4 and 6, Section 3). Such objects will retain their original size and shape (barring wraparound), while objects containing absolute cursor commands may be partially distorted when displayed from a "starting" position different from the position of the cursor when the object was created.

Purpose: To open a Computek graphic object file.

Calling sequence: CALL CKOFIL(LDNF,NEW[,&rcn])

Parameters:

LDNF - either a fullword integer (between 0 and 19) specifying the logical I/O unit to which the file is attached, or a file name followed by one or more blanks.
NEW - a fullword integer one or zero, 1 indicating that the file has been previously initialized and 0 indicating a new, uninitialized file.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - illegal logical device number or file name
8 - read error on attempt to open the file

Description: CKOFIL is used to open a file which is to be used for graphic objects created by means of CKCOB. This routine must be called before using CKLOB, CKSOB, or CKGOB, since it acquires directory information from the file, sets up a FDUB pointer and sets up information necessary for working with sequential files. If the file is a new graphic object file, this routine initializes the file directory and then sets up the information described above. It should be noted here that a graphic object file must be a sequential file; it must also have existed before being referenced by CKOFIL, though it need not have been attached to a logical I/O unit if it is referenced by name.

Purpose: To create a Computek graphic object.

Calling sequence: CALL CKCOB(NAME[,&rcn])

Parameters:

NAME - a fullword containing the name of the object to be defined.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - the program was already in object definition mode
8 - an object of the name specified already exists

Description: CKCOB is used to create a graphic object. A graphic object is constructed by making first a call to CKCOB, then the calls which produce the desired object, and finally the call or parameter which transmits the output buffer. That is, the object is defined as the contents of the output buffer on the first request to transmit after a call has been made to CKCOB. The active output location cannot be a core location during object creation. Note: objects are created region-dependent, but transmission is region independent; thus, region scaling occurs at creation time and not at transmission time.

1 of 1

Purpose: To save one or more objects in the graphic object file that is currently open.

Calling sequence: CALL CKSOB(VNAME,NCNT[,&rcn])

Parameters:

VNAME - a fullword vector of object names.
NCNT - a fullword integer containing the count of the number of names in VNAME.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - an object with the name specified has already been saved
8 - no object with the name specified exists
12 - no file has been opened, or directory overflow

Description: CKSOB is used to save objects (created during the current run) in the graphic object file that is currently open. Note: since this process of saving objects in a file involves updating directory information as well as writing the new object into the file, it is suggested that CKSOB be called as seldom as possible. For example, if 10 objects were created during the run and they were all to be saved, it would be more than ten times as expensive to save the objects by ten separate calls on CKSOB than it would be to call CKSOB once with a vector containing the ten names.

Purpose: To acquire one or more objects from a graphic object file.

Calling sequence: CALL CKGOB(VNAME,NCNT[,&rcn])

Parameters:

VNAME - a fullword vector of object names.
NCNT - a fullword integer containing the count of the number of names in VNAME.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - no object with the name specified is in the directory
8 - no file has been opened

Description: CKGOB is used to acquire--"get"--an object or group of objects from the currently open graphic object file. Objects which have been saved on the file by CKSOB, and are to be used during a run, must be acquired before CKIOB can be used to place them into the output buffer. An object need be acquired only once per run, unless it has been destroyed after having been previously acquired.

Purpose: To insert a Computek graphic object into the output buffer.

Calling sequence: CALL CKIOB(NAME[,IB][,&rc4])

Parameters:

NAME - a fullword containing the name of the object
IB - if present, is a fullword integer specifying whether transmission is desired.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - object with name specified does not exist

Description: CKIOB copies the object named into the current end of the output buffer. The object must have been created, by CKCOB, or acquired, by CKGOB, before insertion.

Purpose: To destroy graphic objects which are no longer needed.

Calling sequence: CALL CKDOB(VNAME,NCNT)

Parameters:

VNAME - a fullword vector containing the name(s) of the object(s) to be destroyed.

NCNT - a fullword integer containing the count of the number of names in VNAME.

Description: CKDOB destroys those objects specified in the vector of names passed. Storage is acquired whenever objects are created; this routine provides a method of releasing that storage when the object is no longer needed. Note: this routine has no effect on graphic object files.

Purpose: To get a list of the names of all objects that have been saved on a graphic object file.

Calling sequence: CALL CKLOB(VNAME,NCNT)

Parameters:

VNAME - a fullword vector where the object names are to be placed.
NCNT - a fullword integer that will contain a count of the number of objects in the list.

Description: CKLOB is used to get a list of all the objects saved on the current graphic object file. Note: the program must provide a vector sufficiently long to contain the desired list.

Input Processing

As indicated in the Input Control section of the Introduction and in the preface to the I/O routine descriptions (2.3.0), this group of routines provide program control for design features specific to the Computek terminal. CKRC and CKRCRT supply the program with the current position of the cursor, either dynamically or following a RETURN, respectively; those two routines will probably be used quite frequently. CKFBTN and CKABTN will be used less frequently, primarily when a single input character is to be entered but is not to be echoed for "printing" on the screen. CKATTN serves an essential function for the internal processing requirements of CKLIB, but it will probably be called infrequently by user programs; of its many optional settings for interrupt processing, the one established by default produces a "normal" termination of program execution.

Purpose: To read the current position of the cursor.

Calling sequence: CALL CKRC(X,Y[,REGN][,&rc4])

Parameters:

X	-	a real*4 where the x-coordinate of the logical address of the cursor is to be stored.
Y	-	a real*4 where the y-coordinate of the logical address of the cursor is to be stored.
REGN	-	if present, a fullword region name that the cursor is to be read relative to.
rcn	-	optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0	-	normal return
4	-	region name specified does not exist

Description: CKRC is used to read the current cursor position, returning its x and y coordinates. The cursor is read relative to the active region, unless a region specified in the call should be used in computing the values of the x and y coordinates.

Purpose: To read a single input character and return either the position of the cursor or the value of the character read.

Calling sequence: CALL CKRCRT(X,Y[,REGN][,&rcn])

Parameters:

X - a real*4 where the x-coordinate of the logical address of the cursor is to be stored.
Y - a real*4 where the y-coordinate of the logical address of the cursor is to be stored.
REGN - if present, a fullword region name that the cursor is to be read relative to.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - region name specified does not exist
8 - first character read for CKRCRT was not RETURN

Description: CKRCRT is used to read the cursor position after the user has positioned the cursor then depressed the RETURN key. It returns the x and y coordinates of the cursor relative to the active region, unless a region name has been specified in the call; then it returns coordinates relative to that region. If the first character sent was <u>not</u> RETURN, the value of X will be the floating-point representation of the ASCII input character (0-127), and the value of Y will be the floating-point representation of an integer from 0 through 6, determined as in CKFBTN.

Purpose: To allow reading of the terminal's function buttons.

Calling sequences: CALL CKFBTN(IFCN)
CKFBTN(IFCN)

Parameters:

IFCN - an integer*4 where the integer representation of the input ASCII character is to be stored.

Description: CKFBTN may be called as a subroutine or referenced as a function; in either case, it will read a single input character from the terminal. The returned value of IFCN will be the integer representation of the ASCII character which was entered (i.e., an integer in the range 0 through 127). The returned value of CKFBTN, when referenced as a function, will be an integer in the range 1 through 6 (according to which of the six function keys was pressed) or an integer 0 if a non-function-key character was the character read. If referenced as a function, CKFBTN must be declared of type integer*4.

Note: The first transmitting key depressed will be read immediately, with no RETURN required. In effect, then, characters which ordinarily exercise a special-purpose function (RETURN, PAGE, attention interrupt, etc.) are temporarily disabled and are read as characters. Intricacies of the interface programs require that the user enter these single-character inputs only when prompted to do so. See Appendix E.

CKABTN

Purpose: To read a single input character in binary form.

Calling sequences: CALL CKABTN(N)
CKABTN(N)

Parameters:

N - an integer*4 where the integer representation of the input ASCII character is to be stored.

Description: CKABTN may be called as a subroutine or referenced as a function; in either case, it will read a single input character from the terminal. The returned value of N will be the integer representation of the ASCII character which was entered (i.e., an integer in the range 0 through 127). CKABTN, when referenced as a function, will also have the same integer value. If referenced as a function, CKABTN must be declared of type integer*4.

Note: The first transmitting key which is depressed will be read immediately, with no RETURN required. In effect, then, characters which ordinarily exercise a special-purpose function (RETURN, PAGE, attention interrupt, etc.) are temporarily disabled and are read as characters. Intricacies of the interface programs require that the user enter these single-character inputs only when prompted to do so. See Appendix E.

Purpose: To trap attention interrupts.

Calling sequence: CALL CKATTN(IAT[,MARK][,&rcn])

Parameters:

IAT - a fullword integer indicating the kind of attention processing desired.
MARK - if present, a fullword integer location required in type-2 attention settings.
rcn - optional statement label(s) for transfer on non-zero return code(s) of n.

Return codes:

0 - normal return
4 - type-3 attention interrupt exit
8 - illegal type of attention setting
12 - stack overflow

Description: CKATTN, used to control attention interrupts, allows several courses of action to be taken. (It should be noted here that this routine is provided because an attention interrupt occurring while the Computek is in graphic mode will be interpreted as graphical information. Since this is an undesirable situation, CKLIB traps all attentions so it can make sure the Computek is in character mode before stopping. CKATTN provides a method of overriding this action if desired.) The following types of attention settings are provided:

Type	Action
0	don't trap any attention interrupts.
1	trap attention interrupts, set the Computek mode to character and call MTS. (This is the default setting.)
2	trap attention interrupts, set MARK to 1 (MARK will be preset to zero on a type-2 call), reset attention trap and continue where processing was stopped.
3	trap attention interrupts, reset attention trap and go to the location specified by statement label rc4.

For these four basic types of attention processing, there are 13 possible values for IAT which indicate how these types are to be utilized. The following list gives the action for each of those values of IAT:

IAT	Action
0	type-0 setting
1	type-1 setting
2	type-2 setting
3	type-3 setting
4	type-0 setting and stack previous setting
5	type-1 setting and stack previous setting
6	type-2 setting and stack previous setting
7	type-3 setting and stack previous setting
8	type-0 setting if not already a type-0 setting
9	type-1 setting if not already a type-1 setting
10	type-2 setting if not already a type-2 setting
11	type-3 setting if not already a type-3 setting
12	pop stack once and use that setting as the new setting
13	pop stack once and use that setting as the new setting; also, if an attention interrupt has occurred since the last setting, take action specified by new type, i.e., force the interrupt

Note: there is a limit of 10 on the number of attention settings which may be stacked.

Utilities

This group of routines is independent of the Computek terminal, i.e., they may be used by programs which do not use the Computek as an output device. They are utilized by many of the Computek subroutines and thus in the library, however their generality is such that they may be utilized in other instances, thus they are being included in this manual.

In general routines which are not Computek dependent will not be added to the library unless, as in this case, they are being utilized by some Computek dependent program.

Purpose: To return the number of arguments in the argument list of the calling subroutine

Calling sequence:

```
N = CKNARG(DUM)
```

Parameters:

DUM - is a dummy argument which is needed to satisfy the FORTRAN function calling convention. It is neither examined nor changed.

Description: This function subroutine is meant to be called from a FORTRAN subroutine. It returns the number of arguments in the argument list of the calling subroutine. Note that CKNARG must be declared INTEGER in the calling subroutine.

Purpose: To do conditional buffer transmission based on an optional transmit code from a FORTRAN subroutine.

Calling sequence:

```
CALL CKTRC(N)
```

Parameters:

N - is the location of a fullword integer specifying the position of the optional transmit code in the argument list of the calling subroutine.

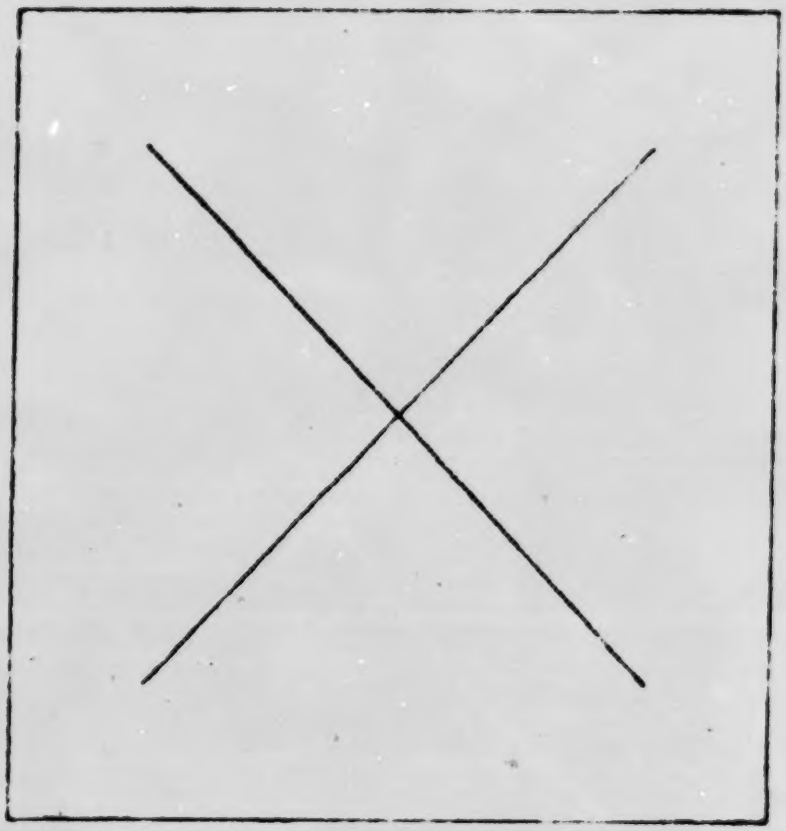
Description: CKTRC is meant to be called from a FORTRAN subroutine. It examines the argument list of the calling subroutine and if the (N - 1)th argument is not the last argument, it transmits the buffer according to the Nth argument. Note that the optional argument must be dimensioned.

Sample Programs and Displays

The examples presented in the following pages consist of two pages each. Pages on the left are photo-reproductions of hard-copies produced by a Tektronix 4601 Hard Copy Unit. Those hard-copies reflect the visual contents of the terminal screen at the time the copy was made. The right-hand page facing each display contains one or more programs whose output produced that particular display. (For pages containing more than one program, one actually produced the copied display and the others produced identical displays.)

These examples are provided for three reasons: (a) to illustrate the general form of FORTRAN programs which call CKLIB routines; (b) to indicate specific calling sequences of some of the more complex routines; and, (c) to permit a comparison of some specific programs with the graphic output they produce. None of the sample programs is lengthy, but some are moderately complex. It is suggested that they be referenced together with the descriptions of those subroutines which they invoke; such cross-referencing can clarify both the specific use of calling parameters and the general capabilities of routines which were used to produce specific displays.

Although the reproduced displays are not as sharp as might be desired, it was felt that they should be included with as close to their actual appearance as possible. The display pages should be viewed with their left edges down; the heavy line at the top of each display page is the result of a similar line which indicates the left boundary of the screen when a hard-copy is made.



3.2

Example 1

```
CALL CKER
CALL CKMA(200,100)
CALL CKVA(200,700)
CALL CKVA(800,700)
CALL CKVA(800,100)
CALL CKVA(200,100)
CALL CKMA(300,200)
CALL CKVA(700,600)
CALL CKMA(300,600)
CALL CKVA(700,200)
CALL CKMA(0,900,1)
END
```

Each of these four programs produces the display opposite. Note the similarity of the top two and of the bottom two. (The CKTXT version does not use the IB vector. Note the equivalence of MA to 0 and VA to 1.)

```
CALL CKER
CALL CKMA(200,100)
CALL CKVI(0,600)
CALL CKVI(600,0)
CALL CKVI(0,-600)
CALL CKVI(-600,0)
CALL CKMI(100,100)
CALL CKVI(400,400)
CALL CKMI(-400,0)
CALL CKVI(400,-400)
CALL CKMA(0,900,1)
END
```

```
      DIMENSION IX(10),IY(10),IB(10)
      DATA IX/200,200,800,800,200,300,700,300,700,0/
      DATA IY/100,700,700,100,100,200,600,600,200,900/
      DATA IB/0,1,1,1,1,0,1,0,1,0/
      CALL CKER
      CALL CKVABS(IX,IY,10,IB,1)
      END
```

```
      DIMENSION IX(10),IY(10),IB(10)
      DATA IX/200,200,800,800,200,300,700,300,700,0/
      DATA IY/100,700,700,100,100,200,600,600,200,900/
      DATA IB/0,1,1,1,1,0,1,0,1,0/
      CALL CKER
      CALL CKTXT('MAVA...4MAVAMAVAMA#1',IX,IY)
      END
```

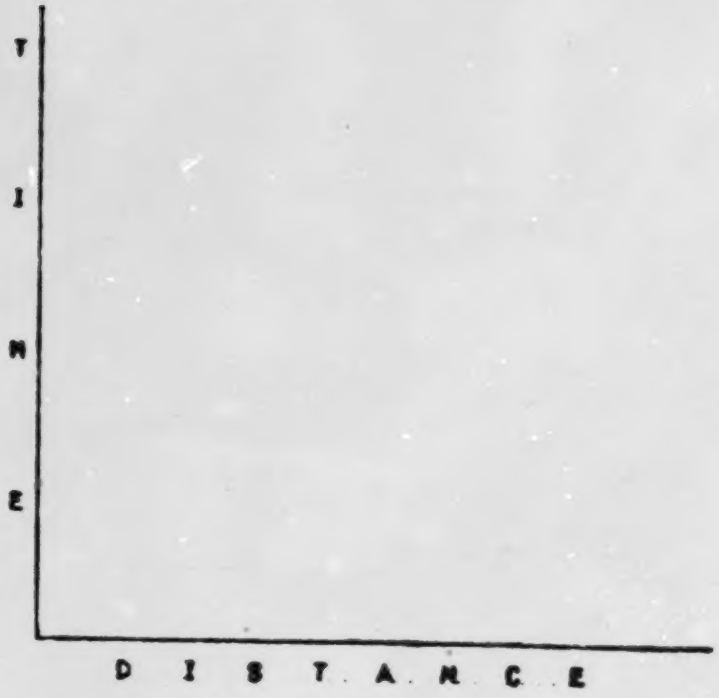
Example 2

```
      CALL CKER
      CALL CKMA(600,700,1)
      WRITE(6,10)
10    FORMAT('ITEM 1')
      CALL CKMA(600,680,1)
      WRITE(6,20)
20    FORMAT('ITEM 2')
      CALL CKMA(600,660,1)
      WRITE(6,30)
30    FORMAT('ITEM 3')
      CALL CKMA(600,640,1)
      WRITE(6,40)
40    FORMAT('ITEM 4')
      CALL CKMA(600,620,1)
      WRITE(6,50)
50    FORMAT('ITEM 5')
      CALL CKMA(0,900,1)
      END
```

Both of these programs produce the display opposite. Note particularly the use of the "B" (base command) in the CKTXT command string; it fixes the point (600,700) as the base point for succeeding line feeds.

```
      DIMENSION COM(13)
      DATA COM/'B"ITEM 1"LF"ITEM 2"LF"ITEM 3"LF"ITEM 4"L',
     C'F"ITEM 5"#1'/
      CALL CKER
      CALL CKTXT(COM,600,700)
      CALL CKMA(0.,900.,1)
      END
```


1 < 1

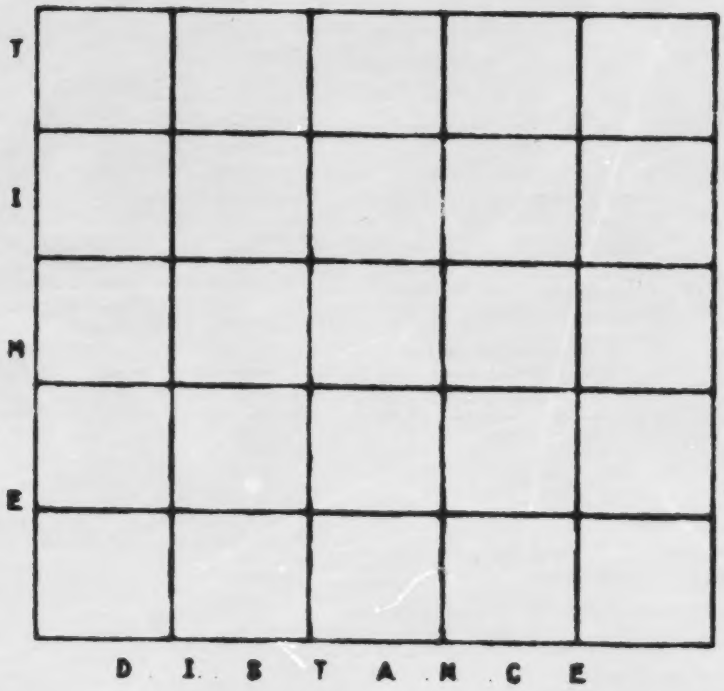


3.6

Example 3

```
DIMENSION X(5),Y(5),COM(10)
DATA X/500.,500.,1000.,480.,560./
DATA Y/700.,200.,200.,660.,170./
DATA COM/'MAVA...2MALRS5"TIME"MALDS3"DISTANCE"#1'/
CALL CKER
CALL CKTXT(COM,X,Y)
CALL CKMA(0.,900.,1)
END
```

The display opposite makes use of "vertical/horizontal print commands" and "space" command of CKTXT to produce attractively formatted labels with a minimum of format information.

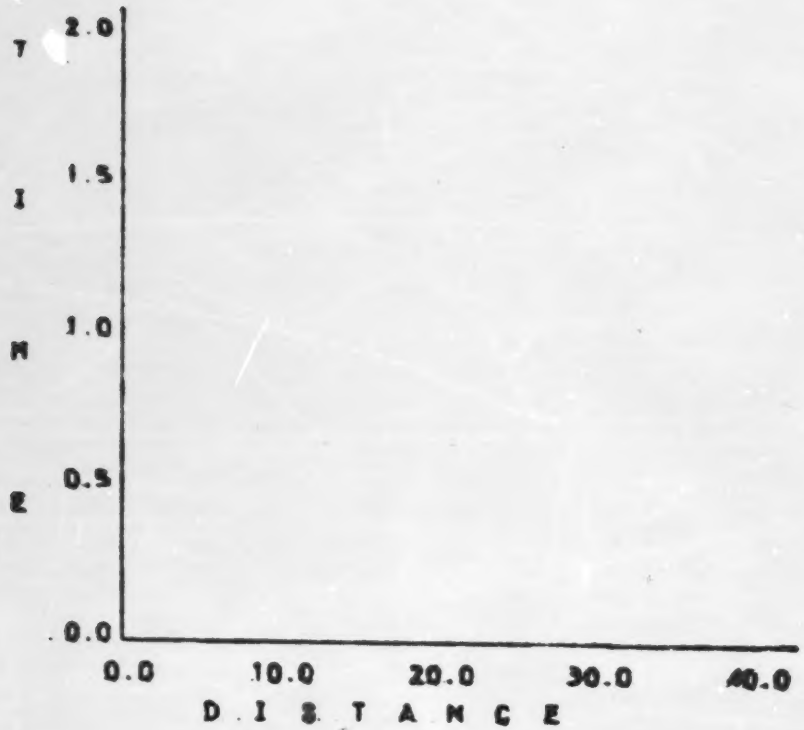


3.8

Example 4

```
      DIMENSION X(7),Y(7),COM(29),IOBJ(2)
      DATA X/500.,600.,700.,300.,900.,480.,560./
      DATA Y/200.,200.,200.,200.,200.,660.,170./
      DATA COM/'MA0101010101MA0101010101MA0101010101',
     C'MA0101010101MA0101010101MALRS5"TIME"MALDS3"DISTANCE"#1'/
      DATA IOBJ/-1,'BOX1'/
      CALL CKCOB('BOX1')
      CALL CKRECT(100.,100.,0)
      CALL CKMI(0.,100.,1)
      CALL CKER
      CALL CKTXT(COM,X,Y,IOBJ)
      CALL CKMA(0.,900.,1)
      END
```

The display opposite was produced by creating a square as a graphic object, then copying that object 25 times with the CKTXT "object insertion" command. Looking at the object definition and the X and Y vectors, it can be seen that the grid was produced as five columns of five squares each, the MA command starting the cursor at the bottom of each column. The labels were produced as in the previous example.



3.10

Example 5

```
      DIMENSION EDGES(10),X(9),Y(9),ISTR(21),COM(30),IF(6)
      DATA EDGES/0.,0.5,1.,1.5,2.,0.,10.,20.,30.,40./
      DATA IF/0,10,4,4,1,0/
      DATA X/500.,500.,1000.,420.,560.,446.,-48.,476.,70./
      DATA Y/700.,200.,200.,660.,140.,200.,120.,170.,0./
      DATA COM/'MAVA...2MALRS5"TIME"MALDS2"DISTANCE"SOMA"<>"MI"<>"MI',
     C'7"<>"MI7"<>"MI7"<>"MA"<>"MI"<>"MI9"<>"MI9"<>"MI9"<>"MI9"<>"#1'/
      CALL CKCVT(EDGES,ISTR(2),IF)
      ISTR(1)=1
      CALL CKER
      CALL CKTXT(COM,X,Y,ISTR)
      CALL CKMA(0.,900.,1)
      END
```

The display opposite was generated by using CKCVT to convert the numeric values in the vector EDGES into character output form. Those character strings were then inserted into the CKTXT command string, separated by cursor movements appropriate for positioning them in the display. The display's axes and labels were produced by commands similar to those of Example 3.

MERRY CHRISTMAS

FROM CRLT

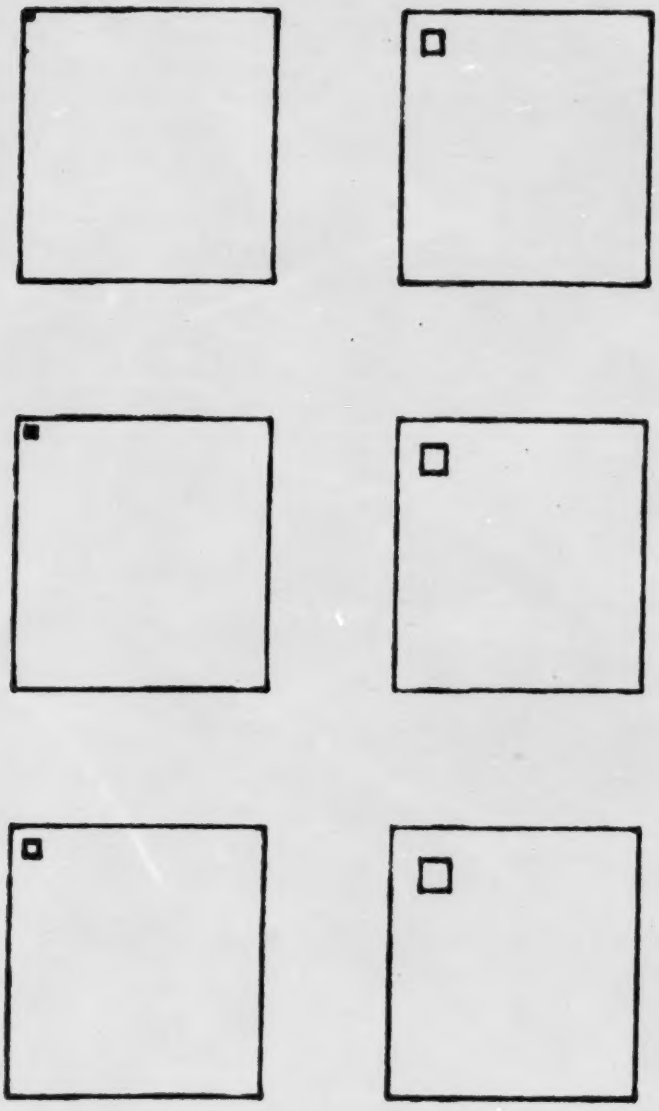
Example 6

```
 INTEGER XSTEM(4)/515,515,485,485/,YSTEM(4)/355,280,280,355/,
*IBSTEM(4)/0,1,0,1/,XSTAR(15)/-12,31,-38,31,-12,0,0,-23,-11,
*20,-6,36,6,3,9/,YSTAR(15)/-36,22,0,-22,38,5,10,-27,3,-31,-8,
*3,-8,36,3/,IBSTAR(15)/5*1,0,1,0,1,0,1,0,1,0,1/,NSTAR/'STAR'/,
*CSTR(10)/'B"MERRY  CHRISTMAS"LFLF"   FROM  CRLT"#0'/
 REAL XTREE(13)/485.,320.,450.,380.,400.,420.,500.,580.,520.,
*620.,550.,680.,515./,YTREE(13)/355.,300.,430.,460.,600.,
*580.,700.,580.,600.,460.,430.,300.,355./,SITREE(12)/.7,.5,.6,
*.7,.5,.7,-6.,.0,-4.,.0,-4.,.0/,SFTREE(12)/.0,4.,.0,4.,.0,6.,
*-.7,-.5,-.7,-.6,-.5,-.7/,X/400./,Y/200./
 CALL CKCOB(NSTAR)
 CALL CKVINC(XSTAR,YSTAR,15,IBSTAR,1)
 CALL CKVABS(XSTEM,YSTEM,4,IBSTEM)
 CALL CKCUSA(XTREE,YTREE,13,SITREE,SFTREE,0)
 CALL CKMA(400,200)
 CALL CKTXT(CSTR,X,Y)
 CALL CKMA(500,727)
 CALL CKIOB(NSTAR)
 CALL CKMA(950,775)
 CALL CKIOB(NSTAR)
 CALL CKMA(0,760)
 CALL CKTRAN
 CALL SYSTEM
 END
```

This program, which illustrates curve generation using CKCUSA, was first compiled into an object file. That object file was then $RUN with K033:CKLIB and with SPUNCH assigned to the file TREE. The display opposite was then produced by entering the system command

$COPY TREE *SINK*@BIN

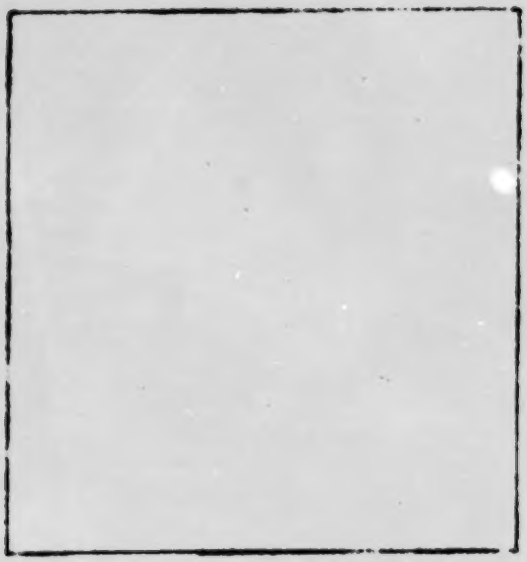
which appears at the upper-left corner of the display.



Example 7

```
      CALL CKER
      CALL CKREG(1,100.,300.,200.,400.)
      CALL CKREG(2,400.,600.,200.,400.)
      CALL CKREG(3,700.,900.,200.,400.)
      CALL CKREG(4,100.,300.,500.,700.)
      CALL CKREG(5,400.,600.,500.,700.)
      CALL CKREG(6,700.,900.,500.,700.)
      DO 10 I=1,6
      CALL CKACT(I)
      CALL CKMA(0.,0.)
10    CALL CKRECT(1023.,1023.,0)
      CALL CKTRAN
      DO 20 I=1,6
      SC=FLOAT(I)
      CALL CKSCA(I,SC,SC)
      CALL CKMA(20.,20.)
20    CALL CKRECT(20.,20.,0)
      CALL CKTRAN
      CALL CKIACT
      CALL CKMA(0.,900.,1)
      END
```

This program consists primarily of three sections: the definition of six logical regions, a loop which activates and outlines each of the regions' boundaries, and a loop which draws a successively growing square within each region. Finally, the base region is reactivated and the cursor moved off the screen.



Example 8

```
      DIMENSION IFORM(6),INSTR(20),X(10),Y(10),IB(10)
      INTEGER*2 LEN
      DATA IFORM/8,1,4,64,0,130/,IB/0,1,1,1,1,1,1,1,1,1/
      CALL CKDER
      CALL SCARDS(INSTR(2),LEN,0,LNUM)
      INSTR(1)=LEN
      CALL CKCVT(J,INSTR,IFORM)
      IFORM(1)=5
      DO 10 I=1,J
      CALL CKNEXI(X(I),1,IFORM)
10    CALL CKNEXI(Y(I),1,IFORM)
      CALL CKVABS(X,Y,J,IB)
      CALL CKMA(0.,900.,1)
      END
```

In the creation of the display opposite, the above program erased the screen and read a line of user input via the system routine SCARDS; that input string, 11 digits each followed by a blank, was read into the vector INSTR with the length of that string going into INSTR(1). The first input item was then converted to an integer value placed in J, the conversion-type was changed, and then J pairs of input items were converted (and scaled) by CKNEXI into real*4 vectors, X and Y. CKVABS then drew the figure specified by that particular input line. Note that the beam vector, IB, was long enough to have accommodated ten input points, rather than the five shown here, and that the "64" which appears in the fourth word of IFORM is the decimal equivalent of the hexadecimal "40"--the code which specifies a blank as the separator character.

The partially illegible input line was "5 1 1 1 5 5 5 5 1 1 1 ".

APPENDICES

4.0

Appendix A: Additional References

Barnes, John. "Manual of Available Software for COMPUTEK 400/20 Display." Offset duplication. Engineering Graphics Laboratory. Ann Arbor: The University of Michigan, 1970.

Computek, Inc. User's Manual: Series 400 CRT Display System. Bulletin 400M. Cambridge, Mass., 1969.

Dertouzos, M. L. "PHASEPLOT: An On-Line Graphical Display Technique." In IEEE Trans. on Elec. Comp. April, 1967.

Dertouzos, M. L. "Character Generation from Resistive Storage of Time Derivatives." In FJCC Proc., vol. 35. Montvale, N. J.: AFIPS Press, 1969.

Halpern, Jeanne W. Computer Graphics. Research News, vol. 21, nos. 4-5. Ann Arbor: The University of Michigan, 1970.

Phillips, R. L. "Storage-Tube Graphics Terminals." In Computing Center Newsletter, vol. 1, no. 10. Ann Arbor: The University of Michigan, 1971.

University of Michigan Computing Center. MTS: The Michigan Terminal System. Third edition. 12 vols. Ann Arbor: The University of Michigan, 1969ff.

Mills, D. L.; DiGiuseppe, J. L.; and Gerstenberger, W. S. Data Concentrator User's Guide. CONCOMP Project Technical Report 28. Ann Arbor: The University of Michigan, 1970.

Appendix B: Basic Operation of the COMPUTEK Terminal

1. Push ON and LINE buttons if necessary--they must be lit; when screen becomes lit (bright green), press ERASE and RESET keys.

2. For 300/150 baud operation (acoustic coupler):

   2a. Set the terminal's rate switch (in back) to "300" and attach the multi-prong data plug to the coupler.

   2b. Turn the acoustic coupler on, dial 3-1500 and insert the telephone handset in the coupler.

   2c. After a pause of a few seconds, a proper connection will be indicated by the printing of characters--probably "fx". Touch the RETURN key.

2. For 1200/150 baud operation (RFL modem):

   2a. Set the terminal's rate switch (in back) to "1200" and attach the multi-prong data plug to the modem; turn the modem on by depressing the push-switch in back (its red light must be on).

   2b. Dial 3-0425 or 3-0426; when the carrier signal is heard (a high-pitched tone), pull up the telephone's white button, wait 2 seconds, and press the terminal's RETURN key (the telephone handset is left off the hook).

3. The system will respond with its recognition message and its "#" prompting character. Touch the SHIFT key and release it (to unlock the shift and get into lower-case), and sign on as you would with a teletype. Pressing the INTER key will suppress the printing of your password.

4. Enter "source k033:setup" (that's kay-zero-three-three) as your first command after sign-on. That file contains device-dependent commands which make COMPUTEK interaction more convenient.

5. Wait for the "SOURCE PREVIOUS" message and the "#" prompt, then proceed with the first command of your session.

6. Remember to turn off both the COMPUTEK and the coupler (or modem) if no one else intends to use the machine when you are through.

Basic Control Functions

Delete-character - - - BACKSPACE	Literal-next - - - - - - - - - CTRL-P
Delete-line - - - - - DEL	Suppress printing - - - - - INTER
End-of-line - - - - - RETURN	Erase screen - - - - - - - - ERASE
End-of-file - - - - - CTRL-C	Reset cursor (upper left) - RESET
Attention Interrupt - CTRL-E	Restore dim screen image - - VIEW

Appendix C: Curve Equations

Each curve (as described on 1.8-1.10) drawn on the screen is the hardware's response to 31 bits of information sent from the system. (That information, of course, is determined by the program and by CKLIB.) Those 31 bits specify where the cursor is to go (from its current position) and the path it is to take in getting there. Although a 32nd bit also is sent, specifying either a visible or "invisible" curve, basic CKLIB routines turn this bit "on" so that all curves are visible. The 31 bits specify:

IX (10 bits) -- x-coordinate (or increment) of the target point;
IY (10 bits) -- y-coordinate (or increment) of the target point;
a_s (1 bit) -- whether the curve will lie above or below the line joining the end-points (i.e., the diagonal of the rectangle they define);
a (5 bits) -- the "target-point" curvature parameter; and,
b (5 bits) -- the "starting-point" curvature parameter.

The function of the IX and IY values is obvious, in view of the 1024x1024 physical address grid. The roles of the remaining three values are not obvious algebraically, but are relatively clear from an intuitive geometric viewpoint. The cursor "knows" its target from the coordinate information. The a_s bit tells it whether to get there by a "northern" or "southern" route, while the a and b parameters tell it how much to curve and how that curvature is to be distributed along its path. In general terms, the latter two values determine the inclination (absolute slope) of the curve away from the diagonal at the curve's two end points.

CKLIB offers the programmer three alternative ways to specify a curve; i.e., three different methods by which various CKLIB routines can translate program values into the above curvature parameters. The easiest to use is CKCUSA, which requires only the coordinates of successive points and the slope of the desired curve as it "leaves" all but the last point and "approaches" all but the first point. Pre-planned displays are particularly well suited for CKCUSA, since all points and slopes are known in advance by the programmer; programs which dynamically change coordinate and slope values may cause trouble for CKCUSA, however, since many combinations of those values cannot be converted into appropriate curvature parameters. (The smooth curve on the cover of this manual, for example, cannot be drawn by the hardware as a single curve, although a reasonable approximation is possible with two curves.)

A second approach to curve generation involves the use of positive normalized slopes, whose values must be computed by the user's program before being sent to CKLIB routines for translation into curvature parameters. The description of CKCS (2.2.17) contains the definition of

"positive normalized slope" in terms of values which will be available to the user's program for testing and modification, if necessary, during program execution. (His program could "recognize", for example, that the curve on the manual cover must require at least one intermediate point, since initial and final slopes are both zero.) In general, of the initial and final slopes, one must be greater than the slope of the diagonal and one must be less, which means that of the two positive normalized slopes, one must lie between 0.0 and 1.0 and the other must be greater than 1.0. (However, if the two positive normalized slopes are equal, the resulting curvature parameters will produce the straight-line diagonal vector, regardless of the slope of the diagonal.) As with CKCUSA, certain combinations of positive normalized slopes cannot be converted into legal curvature parameters for all point-pairs. It is not possible, for example, for the cursor to "leave" the starting point in an almost vertical or horizontal direction (a very large or very small positive normalized slope) and still "arrive" at the end point with a slope very close to that of the diagonal (a positive normalized slope close to 1.0).

The principal shortcoming of these first two methods of curve specification, then, is the possibility that a set of dynamically calculated points and slopes will imply a geometric curve which the hardware cannot produce, at least as a single curve. Should that occur, the called routine will return with an error code of 4, giving the program an opportunity either to modify the original slopes or to add intermediate points for the drawing of a "compound" curve. The need to provide for error processing is the price which the programmer must pay for the convenience of specifying curves by using parameters similar to those of conventional analytic geometry.

The third method of curve specification, on the other hand, allows the program to specify any particular member of the set of curves which the terminal is capable of producing. (As a point of interest, that set is of order approximately 2**50; for given end-points, the number of connecting curves is approximately 2**11.) This method, the one used for calls to CKCA and CKCI (see 2.2.15 and 2.2.16), requires that the program supply two integer values, IA and IB, which are to be translated directly into the three hardware curvature parameters, a_s, a and b. As long as IA and IB are passed with values in the given ranges, valid curvature parameters will always result and the corresponding curve information will be placed in the output buffer. (Note: the resulting curve may be a point or the diagonal vector, if the end-points are the same point or if IA = ±32, respectively.) To gain the convenience of precise and error-free curve specification, however, the calling program must "talk cursor"; that is, must specify curves in terms of the dynamically changing coordinate system of the cursor as it moves from point to point. The geometric effects of the three curvature parameters may be summarized as follows:

(1) if $a_s = 1$, the curve will be drawn above the diagonal; if $a_s = 0$, the curve will be drawn below the diagonal.

(2) as a gets larger (closer to 31), the inclination of the curve at the target point gets closer to that of the diagonal.

(3) as b gets smaller (closer to 0), the inclination of the curve at the starting point gets closer to that of the diagonal.

(4) for a given value of a, increasing values of b will produce curves whose curvatures become more and more abrupt near the target point.

(5) for a given value of b, decreasing the value of a will produce curves whose curvature becomes more and more abrupt near the starting point.

The qualitative effects of the curvature parameters are described more precisely by the analytic representation of the cursor's path as it moves from the starting point to the target point. In the two equations given below, the variable meanings are,

$$\alpha = \frac{a}{31}$$

$$\beta = \frac{1}{0.1 + 0.9/(31.0 * b)}$$

Δx and Δy represent the coordinate distances between starting and target points, and

x and y represent the coordinates of points which lie on the curve, relative to a starting-point value of (0,0).

The curve equations, then, are for $a_s = 1$:

$$\frac{y}{\Delta y} = 1 - \alpha\left(1 - \alpha\frac{x}{\Delta x}\right) - (1 - \alpha)\left(1 - \frac{x}{\Delta x}\right)^{\beta}$$

and, for $a_s = 0$:

$$\frac{x}{\Delta x} = 1 - \alpha\left(1 - \alpha\frac{y}{\Delta y}\right) - (1 - \alpha)\left(1 - \frac{y}{\Delta y}\right)^{\beta}.$$

It can be seen, then, that setting a = 31 (i.e., IA = ±32) will give rise to straight lines, while setting a = 0 (i.e., IA = ±1) will give rise to power curves.

Positive normalized slopes, SI and SF, which are passed to routines CKCS, CKCA and CKCI, are converted into curvature parameters a and b so that the following relations hold:

$$\alpha = SF$$

$$\beta = \frac{SI - SF}{1 - SF} .$$

Users desiring further information on curve specification should consult the User's Manual listed in Appendix A.

Appendix D: Updating Procedures for Manual and Library

The original version of CKLIB was installed on June 30, 1971 and has undergone rather heavy use by the time this first version of the manual is being completed in late August. Six new routines have been added while the manual was in preparation and several of the others have been debugged and/or modified to a certain extent. Those changes have been reflected in the manual in an effort to "begin" with documentation as complete and accurate as practicality would permit. It is to be expected, however, that discrepancies between the manual and CKLIB will arise: new routines will be added, old ones will be modified and previously undetected "bugs" will show up. The following procedures and policies have been established, therefore, to facilitate use of the library without placing an undue burden on the person(s) maintaining it:

(1) No financial responsibility is assumed for costs incurred through the use of CKLIB, whether or not such costs may have arisen because of errors in the library or in the documentation.

(2) A user who detects an error either in the library or in its documentation is encouraged to document the error (as clearly as possible) and bring it to the attention of the person(s) who currently are providing maintenance--at this writing, James W. Conklin of the Center for Research on Learning and Teaching. Users are encouraged first, however, to refer to the section on Error Processing.

(3) Users who have developed, or are in the process of developing, software which appears to be of general use, or exhibits a novel application of the terminal, are encouraged to contact the above-mentioned person(s); it may prove desirable (perhaps, even possible) to add general routines to the library itself and/or to provide some sort of "clearinghouse" for significant developments in terminal usage.

(4) Whenever reasonable, changes made in the library will maintain upward-compatibility; that is, programs which worked properly before the change will continue to do so.

(5) Significant changes in the library will be documented on pages designed for inclusion in this manual.

(6) It is assumed that users of CKLIB will initiate terminal sessions with the command $SOURCE K033:SETUP (see Appendix B). Announcements of significance to CKLIB users will appear between the $SET ECHO=OFF and $SOURCE PREVIOUS messages.

(7) Prudence requires that the authors assume responsibility only for those aspects of terminal usage which either are implied specifically by the initial preparation of K033:CKLIB and this manual or are voluntarily assumed. Source versions of individual routines will be provided at the author's discretion; however, source copies of the entire library will not be distributed.

(8) Comments, suggestions and criticism are welcomed. For the most part, however, additional programming and documentation must be provided by the user desiring it; he may then submit such items for possible inclusion in the library or manual.

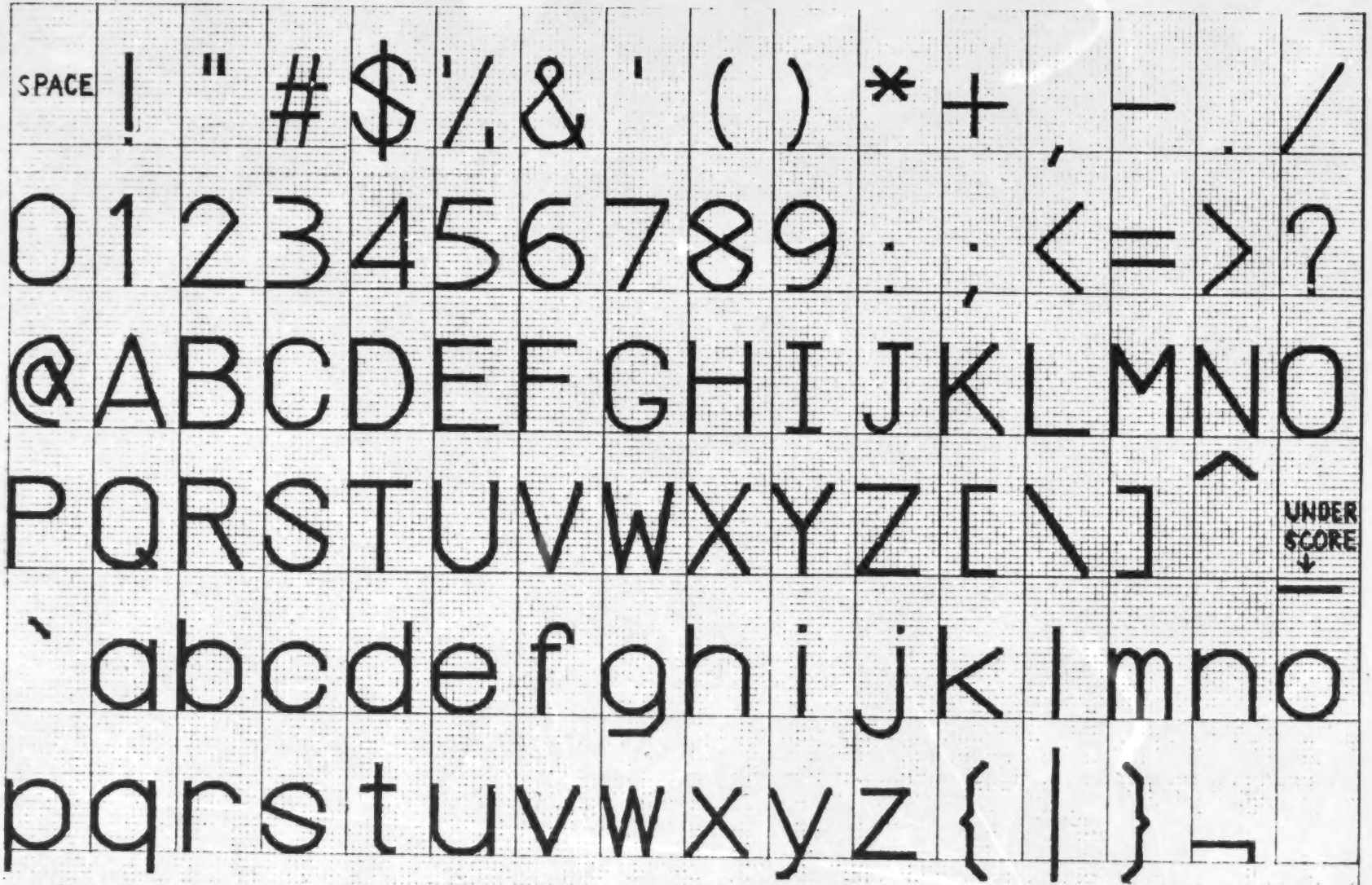
Appendix E: ASCII Characters and Numeric Equivalents

CTRL-@	0	space	32	@	64	`	96
CTRL-a	1	!	33	A	65	a	97
CTRL-b	2	"	34	B	66	b	98
CTRL-c	3	#	35	C	67	c	99
CTRL-d	4	$	36	D	68	d	100
CTRL-e	5	%	37	E	69	e	101
CTRL-f	6	&	38	F	70	f	102
CTRL-g	7	'	39	G	71	g	103
CTRL-h	8	(	40	H	72	h	104
CTRL-i	9	)	41	I	73	i	105
CTRL-j	10	*	42	J	74	j	106
CTRL-k	11	+	43	K	75	k	107
CTRL-l	12	,	44	L	76	l	108
CTRL-m	13	-	45	M	77	m	109
CTRL-n	14	.	46	N	78	n	110
CTRL-o	15	/	47	O	79	o	111
CTRL-p	16	0	48	P	80	p	112
CTRL-q	17	1	49	Q	81	q	113
CTRL-r	18	2	50	R	82	r	114
CTRL-s	19	3	51	S	83	s	115
CTRL-t	20	4	52	T	84	t	116
CTRL-u	21	5	53	U	85	u	117
CTRL-v	22	6	54	V	86	v	118
CTRL-w	23	7	55	W	87	w	119
CTRL-x	24	8	56	X	88	x	120
CTRL-y	25	9	57	Y	89	y	121
CTRL-z	26	:	58	Z	90	z	122
CTRL-{	27	;	59	[	91	{	123
CTRL-|	28	<	60	\	92 |	124
CTRL-}	29	=	61	]	93	}	125
CTRL-¬	30	>	62	^	94	¬	126
CTRL-DEL	31	?	63	_	95	DEL	127

The number in each group's right-hand column is the (base-10) integer value of the bit-code transmitted when the character key in the left-hand column is pressed--for the leftmost group, when the key is pressed while the control key (CTRL) is held down. Those integer values are the ones returned in the arguments of CKFBTN, CKABTN. The codes sent by special keys are as follows:

BACKSPACE	8
TAB	9
LINE FEED	10
PAGE	12
RETURN	13
F1	17
F2	18
F3	19
F4	20
INTER	27
F5	28
F6	29

Note that these special keys duplicate codes which appear in the table. For further details, consult the User's Manual referenced in Appendix A.


SPACE
! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O
P Q R S T U V W X Y Z [\ ] ^
UNDER SCORE
` a b c d e f g h i j k l m n o
p q r s t u v w x y z { | } ~

4.5a

Appendix F: Shape and Placement of Printing Characters

The figure on the page opposite represents a magnification of the standard printing character set of the Computek terminal. Relative sizes, shapes and inter-character spacing are those which would obtain, ideally, if those characters had been displayed in 6 rows of 16 characters each on the terminal screen. The scale is 6.25:1.

As noted in the Introduction, each character cell is 12 physical grid points wide by 20 high, and the beam has a width of one grid cell. These shapes were derived from information contained in the 1969 paper by Dertouzos (see Appendix A).

These cell-by-cell shapes are provided to permit the precise placement of characters on the screen. Terminal hardware generates each character by "starting" the cursor at the lower-left grid point (here, the lower-left cell) of its 12x20 character cell. Thus, for example, a capital "X" could be made to "cross" at the point (x,y) by moving the cursor to the point (x-4,y-6) before displaying the character.

Index

This index is designed to supplement the Table of Contents and the extensive set of cross-references within the text. In general, index terms may appear in the text at locations not indicated here; for ease of reference, only primary occurrences of the terms are listed, and those referenced pages will provide additional reference information.

Active region (see Region)
Address
- Logical - 1.7/1.9/2.1/2.2.0
- Physical - 1.7/1.4/2.2.0
- Transformation - 2.6.0ff/4.5a
- Wraparound - 1.14/1.3

Attention interrupts (see Interrupts)
Base region (see Region)
Beam parameter - 2.2.0/2.2.5
Buffer (see Output buffer)
Character
- Display - 1.1/1.8/2.3.0
- Mode - 1.1
- Set - 1.4/4.5
- Size and shape - 4.5a
- Translation - 2.3.0
- Transmission - 1.7/1.5/4.5

CKLIB - Preface/1.5
CKxxxx (see list at end of Index)
Control characters - 4.2
CTRL key - 1.4/4.5
Cursor - 1.3/1.5/1.8/2.10.0
Curve
- Drawing - 1.9/2.4.0/4.3
- Equations - 4.3ff/2.2.15/2.2.17
- Fitting - 2.4.0
- Normalized slopes - 2.2.17
- Parameters - 4.3ff/2.2.15

Data Concentrator - 1.5/4.1/4.2
Delayed transmission - - - - - - - - - - - - - - - - - - - 2.1
Display - 1.6
Display region (see Region)
ERASE key - 1.5/1.4
Erase-home (also see PAGE key) - - - - - - - - - - - - - - 2.2.1/2.2.2
Error returns - 2.1
Errors - 1.13
Function keys - 1.3/2.10.3/4.5
Function routines - 2.10.3/2.10.4
Graphic object (see Object)
Graphic mode (see Mode)
Grid (see Address)
HOME key - 1.4/1.5
Home-erase (see PAGE key)

Immediate transmission - 2.1/1.11/1.1
INTER key - 1.4/4.2/4.5
Interrupts
    Attention - 2.10.5/4.2
    Program - 1.13
Invisible move - 1.8/2.2.0/2.2.20/4.3
Local control - 1.3
Local mode - 1.3
Logical address (see Address)
Logical region (see Region)
Mode
    Character - 1.1
    Control - 1.3/1.6/1.12
    Graphic - 1.1
Move (also see Beam parameter) - - - - - - - - - - - - - - - 1.8
Object
    File - 2.9.0/2.9.1/1.10
    Graphic - 2.9.0/1.10
Output buffer - 1.11/2.1
Output location - 2.8.0
Overall scale (see Scale factor)
PAGE key - 1.4/1.5/4.5
Parameters (also see Curve) - - - - - - - - - - - - - - - - 2.1
Physical address (see Address)
Program control - 1.5
Region - 2.6.0ff/2.7.0
RESET key - 1.5/1.4
Return codes - 2.1
RETURN key - 1.4/4.2/4.5
Save file - 2.8.0/2.8.5/2.8.6
Scale factor - 2.6.0ff
SETUP file - 4.2/4.4
Slope - 2.2.20/2.2.17
Transmission parameter - - - - - - - - - - - - - - - - - - 2.1/2.9.0
User scale (see Scale factor)
Vector - 1.8/2.1/2.2.0/2.7.0
VIEW key - 1.4/1.5
Visible move - 1.8
Window region (see Region)
Windowing - 2.7.0
Wraparound (see Address)

On the next page is an alphabetical listing of all user routines in CKLIB as of the date of publication. The "CK" prefix has been omitted from the routine names for ease of reference. The single reference given is the first page of the routine's description; most routines are mentioned elsewhere in the text, frequently in the prefaces to each group of routine descriptions.

ACT - - 2.6.5

ARC - - 2.5.3

ARCS - - 2.5.2

ABTN - - 2.10.4

ATTN - - 2.10.5

BLEN - - 2.8.4

CA - - 2.2.15

CERR - - 2.3.3

CI - - 2.2.16

CIR - - 2.5.1

CFIT - - 2.4.2

COB - - 2.9.2

CS - - 2.2.17

CSA - - 2.2.18

CSI - - 2.2.19

CUSA - - 2.2.20

CVT - - 2.3.2

DASH - - 2.5.5

DER - - 2.2.2

DOB - - 2.9.6

DREG - - 2.6.7

ER - - 2.2.1

FBTN - - 2.10.3

GOB - - 2.9.4

GRIN - - 2.6.4

IACT - - 2.6.6

IOB - - 2.9.5

LOB - - 2.9.7

LONG - - 2.2.14

MA - - 2.2.3

MI - - 2.2.6

ML - - 2.2.12

MS - - 2.2.9

NEXI - - 2.3.4

OFIL - - 2.9.1

OPUT - - 2.8.7

ORG - - 2.6.2

PURG - - 2.8.3

RC - - 2.10.1

RCRT - - 2.10.2

RECT - - 2.5.4

REG - - 2.6.1

RSTO - - 2.8.6

SAVE - - 2.8.5

SB - - 2.2.21

SCA - - 2.6.3

SHOR - - 2.2.11

SMTH - - 2.4.3

SOB - - 2.9.3

TRAN - - 2.8.1

TRH - - 2.8.2

TXT - - 2.3.1

VA - - 2.2.4

VABS - - 2.2.5

VABW - - 2.7.1

VFIT - - 2.4.1

VI - - 2.2.7

VINC - - 2.2.8

VL - - 2.2.13

VS - - 2.2.10